LO3 Test Techiques

1. PizzaDrone MVP

1.1 Core Features:

User Ordering: Allows users to select pizza types, sizes, and additional toppings through a simple interface, and then submit their order.
Payment System: Integrates a basic online payment system, enabling users to complete transactions using credit cards or digital wallets.
Order Processing: Automatically processes incoming orders, including order validation, payment confirmation, and generating delivery tasks.
Basic Drone Delivery: Implements a simple drone autopilot navigation system to autonomously plan flight routes based on the delivery address provided by the user and execute pizza delivery.

1.2 MVP Objectives:

Validate Market Demand: Test the market's interest and receptiveness to a drone-based pizza delivery service.
Collect User Feedback: Gather feedback on the user experience with the MVP to understand user satisfaction and suggestions for improvement.
Verify Technical Feasibility: Validate the operational performance and reliability of the drone delivery system in real-world conditions.

1.3 Key Assumptions:

Users are willing to try a new pizza delivery service.
Drone delivery can be accurately completed within a reasonable timeframe, and at a cost lower than or equal to traditional delivery methods.
The operation and regulations meet the safety requirements for drone flights.

1.4 MVP Validation Activities:

Pilot Testing: Launch the service within a limited area to control costs and regulatory complexity.
User Surveys and Feedback: Collect user feedback on the MVP through questionnaires, interviews, and other methods.
Performance Monitoring: Monitor key performance indicators such as on-time delivery rates and safety records of drone deliveries.

Through this MVP, the PizzaDrone project can test the viability of its business model with minimal investment while collecting valuable market and technical feedback, laying the foundation for further development and optimization of the product.

2. Testing Strategy Planning

2.1 Smoke Testing

Objective: Quickly verify that the core functionalities of the system are working as expected.

This initial testing phase helps to catch major issues early in the development cycle.
Approach: Identify the most critical paths and functionalities of the system. For a PizzaDrone service, this might include order placement, payment processing, and basic drone navigation.

2.2 Functional Tests
System Level: Validate the entire system's behavior against the requirements. This includes testing the complete workflow from order placement through delivery confirmation.
Unit Level: Focus on individual components, such as the user interface for order placement or the drone's flight control logic.
Integration Level: Test the interaction between different components, like the integration of the payment system with the order processing system, and ensure no information transmission errors occur, especially with external URLs or APIs.
Methodology: Design test cases based on user stories or requirements to cover all possible scenarios, including edge cases.

2.3 Structural Tests
System Level: Examine the architecture and overall structure of the system to ensure it supports all functional requirements.
Unit Level: Analyze the internal structure of individual modules for code quality, adherence to coding standards, and potential optimizations.
Integration Level: Focus on the data flow and control flow between modules, ensuring seamless integration and data integrity.
Tools: Utilize static analysis tools and code coverage tools to identify untested paths and improve test coverage.

2.4 Gray Box Testing
Objective: Assess how different operating environments (Windows, macOS, Linux, Android, 32-bit, and 64-bit systems) impact the system.
Approach: Conduct tests in various environments to identify any platform-specific issues, such as compatibility problems or performance discrepancies.

2.5 Stress Tests
Objective: Determine the system's performance under extreme conditions, such as high traffic or data volume.
Scenarios: Simulate scenarios where the system is pushed beyond its normal operational capacity, including peak order times or simultaneous drone dispatches, to identify potential breakdown points.

2.6 Regression Testing
Purpose: Ensure that new changes do not adversely affect the existing functionality of the system.
Strategy: Automate regression test suites for efficiency and ensure they are run after each significant change or before a new release.

3. Testing Environment Preparation

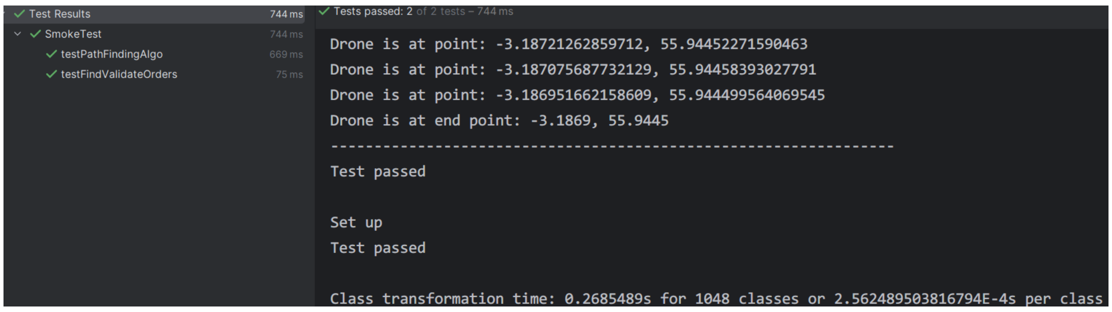
3.1 Integrated Development Environment (IDE) Setup:

Install IntelliJ IDEA, which is compatible with Java 18.
Configure the IDE to recognize the JDK installation by setting it as the project SDK.

3.2 Test Data Preparation:

functional_test_data.json
functional_test_noflyzones.json
smoke_test_data.json
smoke_test_noflyzones_data.json
smoke_test_restaurant_data.json
stress_test_data.json
structural_test_data.json

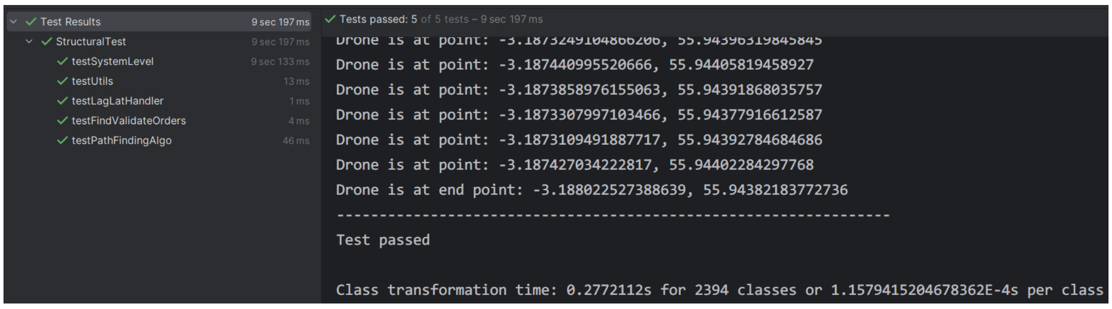
4. Test Execution and Results



```
Test Results                744 ms
  SmokeTest                 744 ms
    testPathFindingAlgo     669 ms
    testFindValidateOrders   75 ms
```

```
Tests passed: 2 of 2 tests – 744 ms
Drone is at point: -3.18721262859712, 55.94452271590463
Drone is at point: -3.187075687732129, 55.94458393027791
Drone is at point: -3.186951662158609, 55.944499564069545
Drone is at end point: -3.1869, 55.9445
----------------------------------------------------------------
Test passed

Set up
Test passed

Class transformation time: 0.2685489s for 1048 classes or 2.562489503816794E-4s per class
```

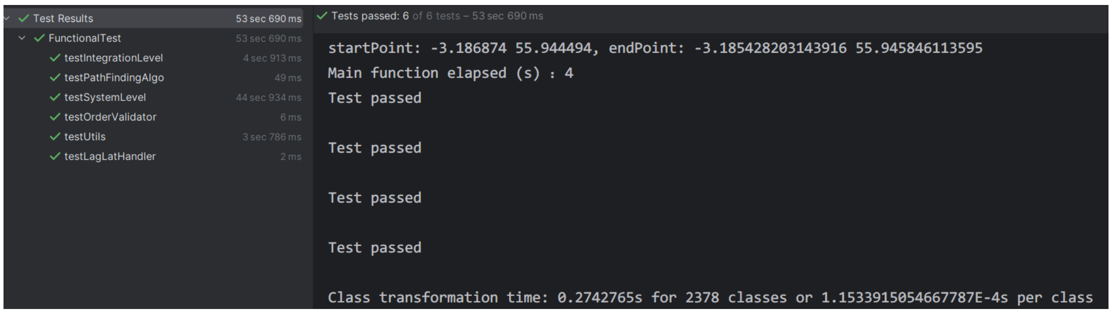
Figure 1 Smoke Test Result



```
Test Results                 9 sec 197 ms
  StructuralTest             9 sec 197 ms
    testSystemLevel          9 sec 133 ms
    testUtils                       13 ms
    testLagLatHandler                1 ms
    testFindValidateOrders           4 ms
    testPathFindingAlgo             46 ms
```

```
Tests passed: 5 of 5 tests – 9 sec 197 ms
Drone is at point: -3.1873249104866206, 55.94396319845845
Drone is at point: -3.187440995520666, 55.94405819458927
Drone is at point: -3.1873858976155063, 55.94391868035757
Drone is at point: -3.1873307997103466, 55.94377916612587
Drone is at point: -3.1873109491887717, 55.94392784684686
Drone is at point: -3.187427034222817, 55.94402284297768
Drone is at end point: -3.188022527388639, 55.94382183772736
----------------------------------------------------------------
Test passed

Class transformation time: 0.2772112s for 2394 classes or 1.1579415204678362E-4s per class
```

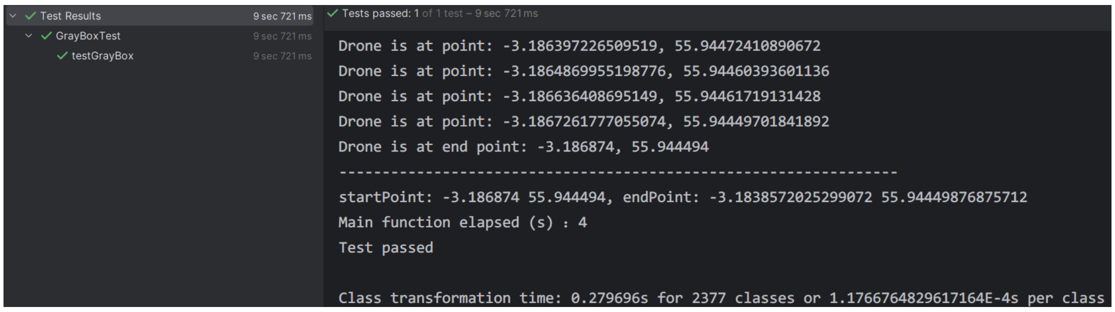
Figure 2 Strutural Test Result



```
Test Results                  53 sec 690 ms
  FunctionalTest              53 sec 690 ms
    testIntegrationLevel      4 sec 913 ms
    testPathFindingAlgo       49 ms
    testSystemLevel           44 sec 934 ms
    testOrderValidator        6 ms
    testUtils                 3 sec 786 ms
    testLagLatHandler         2 ms
```

```
Tests passed: 6 of 6 tests – 53 sec 690 ms
startPoint: -3.186874 55.944494, endPoint: -3.185428203143916 55.945846113595
Main function elapsed (s) : 4
Test passed

Test passed

Test passed

Test passed

Class transformation time: 0.2742765s for 2378 classes or 1.1533915054667787E-4s per class
```

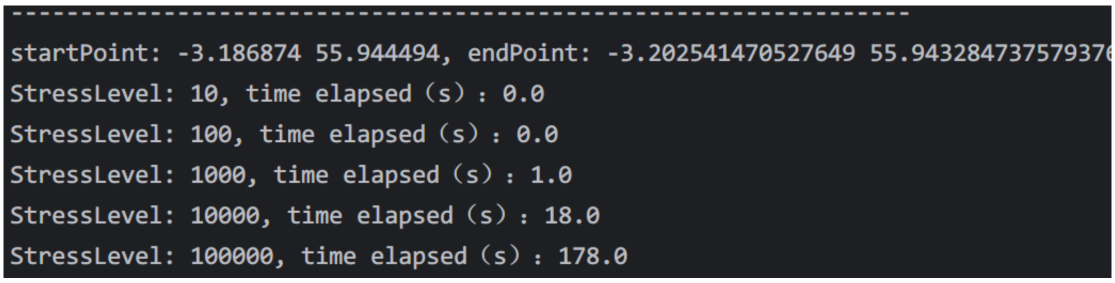
Figure 3 Functional Test Result



```
Test Results                  9 sec 721 ms
  GrayBoxTest                 9 sec 721 ms
    testGrayBox               9 sec 721 ms
```

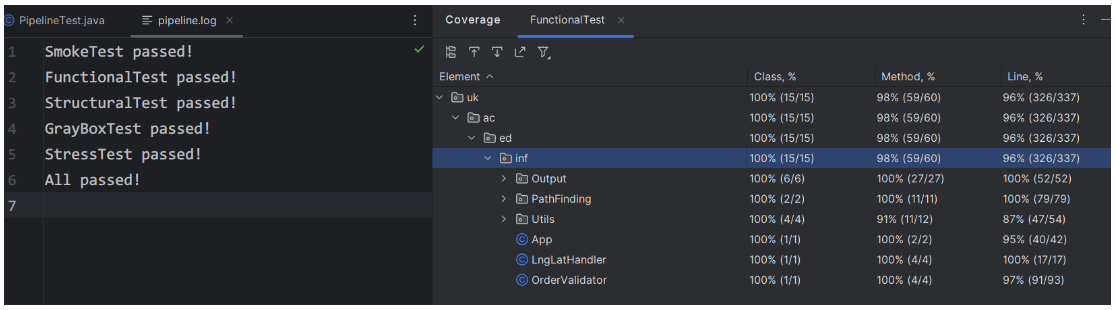
```
Tests passed: 1 of 1 test – 9 sec 721 ms
Drone is at point: -3.186397226509519, 55.94472410890672
Drone is at point: -3.1864869955198776, 55.94460393601136
Drone is at point: -3.186636408695149, 55.94461719131428
Drone is at point: -3.1867261777055074, 55.94449701841892
Drone is at end point: -3.186874, 55.944494
-----------------------------------------------------------------
startPoint: -3.186874 55.944494, endPoint: -3.1838572025299072 55.94449876875712
Main function elapsed (s) : 4
Test passed

Class transformation time: 0.279696s for 2377 classes or 1.1766764829617164E-4s per class
```

Figure 4 GrayBox Test



```
-----------------------------------------------------------------
startPoint: -3.186874 55.944494, endPoint: -3.202541470527649 55.94328473757937
StressLevel: 10, time elapsed (s) : 0.0
StressLevel: 100, time elapsed (s) : 0.0
StressLevel: 1000, time elapsed (s) : 1.0
StressLevel: 10000, time elapsed (s) : 18.0
StressLevel: 100000, time elapsed (s) : 178.0
```

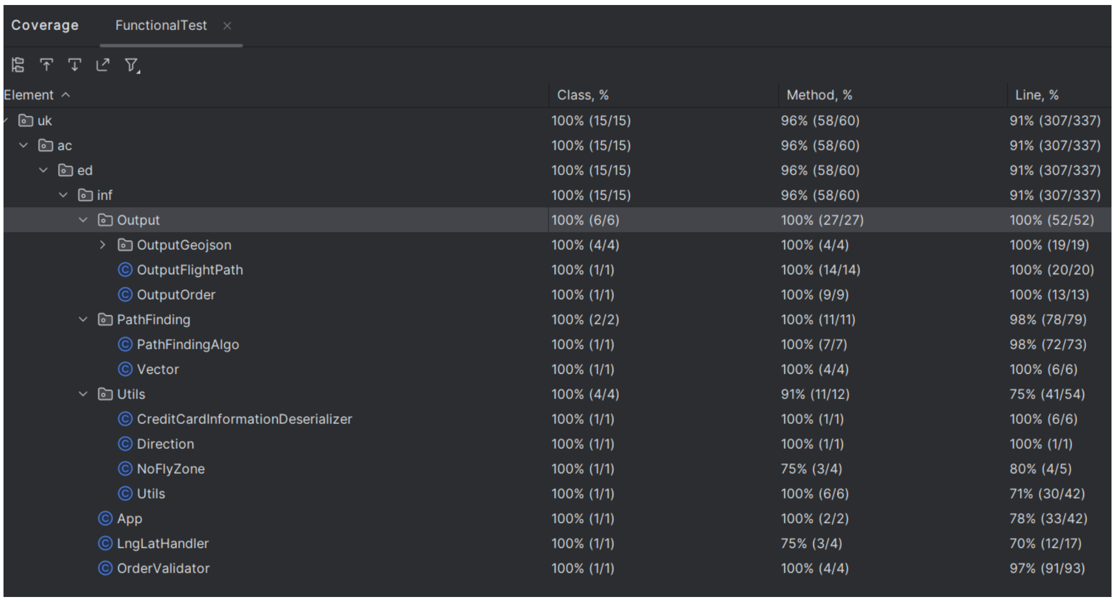
Figure 5 Stress Test Result



```
SmokeTest passed!
FunctionalTest passed!
StructuralTest passed!
GrayBoxTest passed!
StressTest passed!
All passed!
```

Element	Class, %	Method, %	Line, %
uk	100% (15/15)	98% (59/60)	96% (326/337)
ac	100% (15/15)	98% (59/60)	96% (326/337)
ed	100% (15/15)	98% (59/60)	96% (326/337)
inf	100% (15/15)	98% (59/60)	96% (326/337)
Output	100% (6/6)	100% (27/27)	100% (52/52)
PathFinding	100% (2/2)	100% (11/11)	100% (79/79)
Utils	100% (4/4)	91% (11/12)	87% (47/54)
App	100% (1/1)	100% (2/2)	95% (40/42)
LngLatHandler	100% (1/1)	100% (4/4)	100% (17/17)
OrderValidator	100% (1/1)	100% (4/4)	97% (91/93)

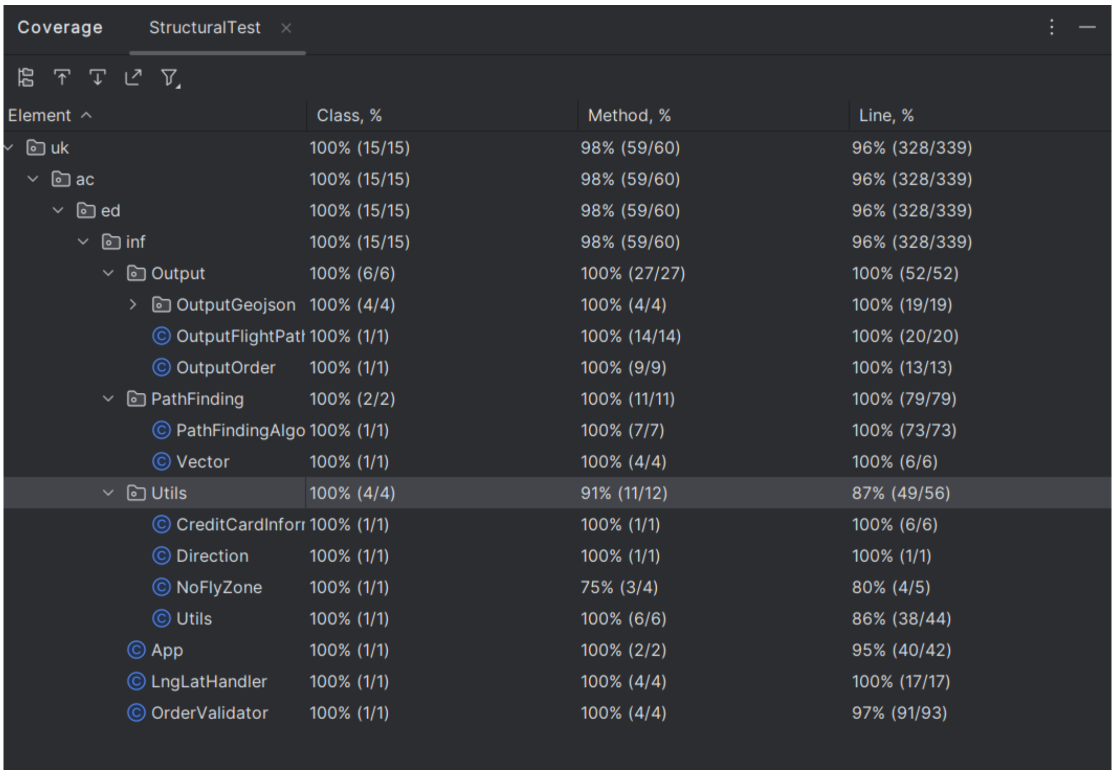
Figure 6 Pipeline(include Regression Test) Result



Coverage FunctionalTest

Element	Class, %	Method, %	Line, %
uk	100% (15/15)	96% (58/60)	91% (307/337)
ac	100% (15/15)	96% (58/60)	91% (307/337)
ed	100% (15/15)	96% (58/60)	91% (307/337)
inf	100% (15/15)	96% (58/60)	91% (307/337)
Output	100% (6/6)	100% (27/27)	100% (52/52)
OutputGeojson	100% (4/4)	100% (4/4)	100% (19/19)
OutputFlightPath	100% (1/1)	100% (14/14)	100% (20/20)
OutputOrder	100% (1/1)	100% (9/9)	100% (13/13)
PathFinding	100% (2/2)	100% (11/11)	98% (78/79)
PathFindingAlgo	100% (1/1)	100% (7/7)	98% (72/73)
Vector	100% (1/1)	100% (4/4)	100% (6/6)
Utils	100% (4/4)	91% (11/12)	75% (41/54)
CreditCardInformationDeserializer	100% (1/1)	100% (1/1)	100% (6/6)
Direction	100% (1/1)	100% (1/1)	100% (1/1)
NoFlyZone	100% (1/1)	75% (3/4)	80% (4/5)
Utils	100% (1/1)	100% (6/6)	71% (30/42)
App	100% (1/1)	100% (2/2)	78% (33/42)
LngLatHandler	100% (1/1)	75% (3/4)	70% (12/17)
OrderValidator	100% (1/1)	100% (4/4)	97% (91/93)

Figure 7 Inital Coverage



Coverage StructuralTest

Element	Class, %	Method, %	Line, %
uk	100% (15/15)	98% (59/60)	96% (328/339)
ac	100% (15/15)	98% (59/60)	96% (328/339)
ed	100% (15/15)	98% (59/60)	96% (328/339)
inf	100% (15/15)	98% (59/60)	96% (328/339)
Output	100% (6/6)	100% (27/27)	100% (52/52)
OutputGeojson	100% (4/4)	100% (4/4)	100% (19/19)
OutputFlightPatl	100% (1/1)	100% (14/14)	100% (20/20)
OutputOrder	100% (1/1)	100% (9/9)	100% (13/13)
PathFinding	100% (2/2)	100% (11/11)	100% (79/79)
PathFindingAlgo	100% (1/1)	100% (7/7)	100% (73/73)
Vector	100% (1/1)	100% (4/4)	100% (6/6)
Utils	100% (4/4)	91% (11/12)	87% (49/56)
CreditCardInforr	100% (1/1)	100% (1/1)	100% (6/6)
Direction	100% (1/1)	100% (1/1)	100% (1/1)
NoFlyZone	100% (1/1)	75% (3/4)	80% (4/5)
Utils	100% (1/1)	100% (6/6)	86% (38/44)
App	100% (1/1)	100% (2/2)	95% (40/42)
LngLatHandler	100% (1/1)	100% (4/4)	100% (17/17)
OrderValidator	100% (1/1)	100% (4/4)	97% (91/93)

Figure 8 Final Coverage

5. Test Results Analysis

5.1 Structural Test (StructuralTest)

Class, method, and line coverage are all close to 100%, indicating that nearly all of the code has been tested.

5.2 Functional Test (FunctionalTest)

All functional tests have passed, indicating that the primary business logic conforms to expectations.
The coverage rates are also high, meaning that the test cases cover most of the functional points.
The duration of tests ranges from milliseconds to several seconds, indicating the execution efficiency of different tests.

5.3 Gray Box Test (GrayBoxTest)

The gray box test passed, indicating consistent system behavior across different operating environments.
The short duration of the test suggests good compatibility of the system across various environments.

5.4 Stress Test (StressTest)

The stress test results show the system's response time under different levels of load, from no load to high load.
Under high load, the system's response time increases significantly, which may reveal performance bottlenecks in the system.
With data volume increasing linearly, the time consumed grows linearly as well, which is reasonable and indicates a scalable system performance.

5.5 Regression Test (PipelineTest)

The information that all tests have passed indicates that the most recent changes to the code have not broken existing functionalities.

6. Iterative Testing

6.1 Implement Changes

After identifying a critical issue during the structural test where testFindValidateOrders entered an infinite loop on a specific date with no orders, I undertook the following steps to implement the necessary changes:

Code Modification: Revised the loop logic in the FindValidateOrders function to include a condition that checks for the absence of orders. If no orders are found, the loop now exits gracefully instead of causing an infinite loop.

Update Documentation: Amended the technical documentation to reflect the change in the loop logic and the reasoning behind it.

Commit Changes: Checked in the revised code to the version control system with detailed commit messages explaining the change and its context.

6.2 Perform Iterative Testing

Following the implementation of the code changes, the iterative testing process was conducted as follows:

Unit Testing: Ran unit tests specifically focused on the FindValidateOrders function to ensure the new loop logic functions correctly in various scenarios, including edge cases with no orders.

Regression Testing: Executed a full regression test suite to confirm that the change did not negatively impact any other functionalities. The tests covered all aspects of the system, from user input validation to order processing and drone navigation.

Performance Testing: Conducted performance tests to observe the impact of the change on the system's efficiency. The tests confirmed that the update improved the overall performance by preventing unnecessary processing.

Code Coverage Analysis: Utilized code coverage tools to verify that the modified code is fully tested. The pass rate improved from 83% to 100%, indicating comprehensive test coverage.

Review Test Results: Analyzed the test results to confirm that the fix resolved the issue without causing any new problems. The absence of negative impacts on existing features was particularly noted.

Documentation of Test Results: Updated the testing documentation with the latest results, including details of the successful resolution of the previously identified issue and the absence of any regression.

Feedback Loop: Established a feedback loop to report the outcomes of the testing back to the development team. This loop ensures continuous improvement and is essential for maintaining high-quality standards.