3. Apply a wide variety of testing techniques and compute test coverage and yield
according to a variety of criteria

3.1 Range of Techniques

A variety of testing techniques were applied to validate the software, including:

- **Unit Testing**: Individual components like OrderValidation, LngLat, and CompassDirection were tested in isolation to ensure they function correctly. For example, the nextStep() method in LngLat was unit-tested to verify it correctly computes the appropriate directional movement.
- **Integration Testing**: End-to-end testing was performed on DeliveryController.calcDeliveryPath() to verify that valid orders correctly produce a flight path while invalid orders return a 400 response. This also ensured proper interaction between OrderValidation, Restaurant, NoFlyZone, and CentralArea.
- **Boundary Testing**: The drone's ability to navigate within the CentralArea and avoid NoFlyZones was tested by using edge case coordinates. Specifically, we tested whether the intersectsNoFlyZone() method in DeliveryController correctly detects intersections with defined no-fly zones.
- **Performance Testing**: The first execution of calcDeliveryPath() was identified as taking over **400ms**, but subsequent runs were significantly faster. This suggested the need for caching or optimizing API calls to reduce the initial request time.
- **Property-Based Testing**: Given the 16 predefined compass directions in CompassDirection, we validated that every movement step conforms to one of these 16 values by running a property-based test over random inputs.
- **Mutation Testing**: To evaluate the robustness of our test suite, mutation testing was applied by introducing small changes to the logic (e.g., modifying angle calculations in nextStep()) and ensuring existing tests caught these deviations.

3.2 Evaluation Criteria for the Adequacy of the Testing

To measure the effectiveness of our tests, we considered:

- **Code Coverage**: We used Intelijis code coverage calculator to measure test coverage. The results indicated high coverage in OrderValidation and LngLat, but slightly lower coverage in pathfinding logic due to untested edge cases involving adjustDirection().
- **Bug Discovery Rate**: Multiple test iterations helped uncover and resolve issues like, The drone making unnecessary detours due to incorrect adjustDirection() logic. The drone occasionally selecting an EAST movement in adjustDirection(), even when a better option existed.
- **Boundary Condition Handling**: Tests ensured the drone never exited the CentralArea once inside and properly avoided NoFlyZones. Failures in this regard led to refining the calculatePath() logic.

3.3 Results of Testing

The testing process identified several key issues:

- **Incorrect Path Adjustments**: The initial path calculation sometimes made unnecessary diversions, leading to inefficient routes. This was resolved by refining adjustDirection() to prioritize the closest alternative direction.
- **Slow Initial Execution**: The first API call to fetch restaurants and no-fly zones took excessive time. Implementing caching improved response times significantly.
- **Inconsistent No-Fly Zone Detection**: Edge cases revealed that intersectsNoFlyZone() sometimes failed to detect minor intersections, requiring adjustments to the intersection-checking algorithm.
- Additionally all 600 unit tests provided from the Orders API were passed:

```
Order No: 392D5CBA
Actual Order Status: VALID
Expected Order Status: VALID
Actual Validation Code: NO_ERROR
Expected Validation Code: NO_ERROR
Order Status and Validation Code Test Passed!
----
Test Summary:
Tests Passed: 600
Tests Failed: 0
All tests passed!

Process finished with exit code 0
```

3.4 Evaluation of the Results

- **Strengths of the Testing Process:**
  - The test suite successfully identified and fixed major issues in navigation, response time, and order validation.
  - Code coverage was comprehensive, covering most critical functions.
  - The drone's movement logic was verified against real-world scenarios.
- **Areas for Improvement:**
  - Additional performance optimizations could further reduce initial execution time.
  - More extensive mutation testing could be performed to improve test robustness.
  - Stress testing with extreme edge cases (e.g., hundreds of no-fly zones) could be conducted.

By employing these testing techniques and refining our testing strategy based on real-world test results, we significantly improved the accuracy and efficiency of the drone delivery system.