## LO4 Limitations of Testing

## Limitations of the Testing Process

**Coverage of all Real-Data Scenarios**
While functional, combinatorial, structural, and model-based testing techniques provide good coverage, it is impossible to simulate every possible scenario that could happen. For instance: Unforeseen environmental factors, such as the project was designed for the local area around Appleton tower if the project is used on a wider range, it may not work correctly e.g. the A star algorithm will take exponentially longer and the 60 second limit will be exceeded.

**Limited Test Data**
The test data used during development is often mock made by the tester, Real data (e.g., order volumes, customer locations, or different restaurants) may introduce edge cases that were not considered or found during testing.

**Dependency on Mocking and Simulation**
Due to the unavailability of the REST server, mocking was used to replicate data. While this enables testing, it lacks the robustness of real integration testing, which could reveal issues in handling live data for example, issues in the API.

**Constraints on Time and Resources**
The 60-second runtime constraint for planning the drone's flightpath limits the extent of testing that can be performed within the application itself. Each test to check this is satisfied would take a while and so the overall tests would take too long to run so the number of these tests had to be limited.

**Statistical Gaps in Test Results**
The code coverage revealed areas that have not been tested in the project and due to time constraints on the testing process not all paths of all methods could be evaluated and even if they are they might not have been tested on a varied case of scenarios/test data

## Review Process

The following are the steps taken in the review process:

**Test Plan Reviews**
The initial test plan was reviewed against the requirements to check it aligned. Reviews were done to validate the coverage of all requirements.

**Code Reviews**
Code reviews were done to find areas of the system that could be potential issues, e.g complex algorithms (FlightAStar) or data handling (FileHandler). Review criteria included: Clearness and readability of the code, if it is following coding standards and the potential for unexpected exceptions or edge cases.

**Coverage**
Code coverage reports were created using IntelliJ inbuilt code coverage after each test run. These reports highlight untested branches, methods and classes in red in IntelliJ which clearly shows where the development of more tests to fill gaps is needed.

## Summary of Findings

The evaluation revealed that the testing process is effective in identifying and addressing many functional and structural issues in the system. However, several limitations remain:

**Uncovered Code**: Some parts of the codebase, particularly edge cases in complex algorithms, were not covered due to limitations in time.

**Dependence on Mocking**: The reliance on simulated data for REST API means that real-world integration scenarios remain untested.

**Resource Constraints**: The amount of time that would be needed for certain tests, such as exhaustive path validation, is not feasible.

To address these limitations, future improvements could focus on:

- Extending test coverage to 100%
- Using real-world data and live system testing if the REST API were to be put back up.
- Look at how the project behaves under stressful conditions, such as high order volumes or unexpected interruptions.

By addressing these limitations, the testing process can achieve higher reliability and robustness, ensuring that the project meets its performance and quality objectives.