LO1: Analyze requirements to determine appropriate testing strategies

1.1 Range of Requirements

Non-functional Requirements:

- Measurable:
    - The program must process fetched orders in under 60 seconds, ensuring efficiency during high demand periods.
    - The flight path calculation for each order should complete within 500 milliseconds.
- Qualitative:
    - Implement robust exception handling to ensure program stability under unexpected conditions (e.g., invalid inputs or server downtime).
    - Provide user-friendly error messages to enhance the user experience in case of failures.
- Data Collection:
    - The program must not store any user data locally or on external servers, ensuring user privacy.
    - Dynamically fetched data (e.g., orders, no-fly zones) should not be aggregated or retained beyond its immediate use.

Functional Requirements:

- Validation of Input Arguments:
    - Ensure input arguments include valid date and REST server URL when program runs.
- Dynamic Data Retrieval:
    - Retrieve data from the REST API, including orders, restaurant locations, and no-fly zones.
- Comprehensive Order Validation Criteria:
    - Payment Card Checks: Validate the 16-digit card number, 3-digit CVV, and non-expired expiry date.

    - Restaurant Validation: Confirm the restaurant exists and is open at the time of the order.
    - Basket Details: Verify the basket includes:
        - An 8-character hexadecimal order number.
        - Pizzas all sourced from the same restaurant.
        - A minimum of 1 and a maximum of 4 pizzas.
        - A valid total cost, including delivery fees.
- Optimal Flight Path Calculation:
    - Calculate the most efficient flight path for each order, starting at Appleton Tower (3.186874, 55.944494) and adhering to these constraints:
        - Follow the 16-wind compass rose.
        - Limit movements to 0.00015° or less.
        - Hover for one move upon reaching the destination.
        - Avoid exiting the central campus area more than once per trip leg.
        - Strictly avoid no-fly zones.
- Generation of Summary Files:
    - deliveries-YYYY-MM-DD.json: Include properties such as orderNo, orderStatus, orderValidationCode, and costInPence.
    - flightpath-YYYY-MM-DD.json: Detail drone movements with properties like orderNo, fromLongitude, fromLatitude, angle, toLongitude, and toLatitude.
    - drone-YYYY-MM-DD.geojson: Provide a visual representation of drone trips in GeoJSON format.

1.2 Level of Requirements

System Requirements:

- Validate the program's ability to:
    - Fetch dynamic data (orders, restaurant locations, and no-fly zones).
    - Perform comprehensive order validation.
    - Accurately calculate flight paths under specified constraints.

    - Generate output files adhering to the required formats.
- Ensure the system responds effectively to valid and invalid inputs, demonstrating seamless end-to-end functionality.

Integration Requirements:

- Confirm the REST API retrieves and integrates order, restaurant, and no-fly zone data correctly.
- Verify that the validation, pathfinding, and file generation components collaborate smoothly.

Unit Requirements:

- Validate specific methods, including:
    - Payment card validation.
    - Flight path calculation.
    - Output file generation.
- Test error handling for incorrect input formats to ensure reliability.

1.3 Identifying Test Approach for Chosen Attributes

Functional Testing:

- Validate input arguments, such as date and REST server URL, to ensure correct handling.
- Test REST API responses to confirm accurate data retrieval.

Performance Testing:

- Measure runtime efficiency for order processing (target: <60 seconds) and flight path calculation (target: <500ms).

Robustness Testing:

- Simulate error scenarios, such as:
    - Invalid data formats for orders or REST responses.
    - Server downtime or unavailability.