Outline of the software being tested:

PizzaDronz: The aim of the software is to read the data about the pizza orders from the server for the specific date, validate the orders and construct paths for the drone to deliver as many valid orders as possible within that day.

Requirements:

System

- SR-1: The drone must aim to deliver >=70% of daily valid orders. (QUALITY)
- SR-2: The system must produce 3 files deliveries-date.json, drone-date.json, flightpath-date.json by the end of the execution. (FUNCTIONAL)
- SR-3: The total runtime of the program for one specified date must not exceed 60 seconds. (PERFORMANCE)
- RS:4 The program must terminate with a specific error code in the event of an error relating to the date, URL and parsing. (ROBUSTNESS)

Integration

- IR-1: The data about restaurants and orders for the specified date must be accessed from the rest server. (FUNCTIONAL)
- IR-2: Orders must be validated based on the: payment info, date, pizzas being from the same restaurant, pizza count between 1 and 4 in the order, and correct calculated cost. Appropriate outcome status must be set for each order, depending on which if any validations did not pass. Helps to ensure that in future only valid orders are delivered. (FUNCTIONAL)
- IR-3: The system must download from the server and store files locally containing data about the central area and no-fly zones. (FUNCTIONAL)

Unit

- UR-1: At the start, the system must be supplied with a server address and a valid date to deliver the orders. (FUNCTIONAL)
- UR-2: The drone cannot leave the central area once it has the order and entered it. (FUNCTIONAL)
- UR-3: The drone cannot fly over no-fly zones. (FUNCTIONAL)
- UR-4: The drone must fly only in one of the 17 given directions. (FUNCTIONAL)

- UR-5: The total number of valid orders for each day must be 40. (QUALITY)
- UR-6: The drone must take on delivery only when it has sufficient battery level for the journey. (ROBUSTNESS)
- UR-7: During the execution of the flightpath, the number of ticks must be recorded for every new move the drone performs. Each new tick is strictly greater than the previous one. (FUNCTIONAL)

Testing Plan

From the broad range of requirements described above, we will choose two of them to construct the testing plan for and go into more detail about them. These requirements are SR-1, a system-level requirement, and IR-2, an integration-level requirement. The software development lifecycle being used here is V-Model.

Below we will look at each requirement and perform a brief assessment of its A&T needs:

- SR-1: This is a measurable quality attribute therefore we need means to measure it. Furthermore, the requirement is not a high priority, so we will not spend a lot of time on it:
    - This requirement can only be tested when the system has been completed, therefore this is a system-level test which most likely occurs towards the end of the lifecycle, namely at the system testing stage.
    - We need to make sure that validation and verification issues are eliminated as much as possible here.
    - To verify, we would need some synthetically generated data and methods to test on the data.
    - To validate, we will need a method of logging how the system performed relative to the quality goal specified in the requirement.
    - This means there is a set of tasks needed to be scheduled for testing:
        - Generating the synthetic data to test the drone performance in terms of delivering the orders.
        - Building scaffolding to create a way of simulating the drone would make it possible to test early on the synthetic data.
        - Design a logging system to record the performance on the data.
        - Design an analytics system to record what affects the performance.
        - Feed the collected information into the early stages of testing.

- IR-2: This requirement is a higher priority as it ensures that only valid orders are delivered and delivered correctly. This means:

  - The requirement can be tested when components responsible for different parts of validations as well as orders parsing components are complete, which suggests that it is an integration-level test. Which is performed at some point late during the lifecycle. In our case, it will be performed somewhere around the middle stage of the lifecycle, namely at the integration testing stage.
  - It must be ensured that each element of validation works properly
  - Make sure it is sensitive and fails all the tests where orders are expected to be invalid.
  - Remove any validation and verification risks as early as possible.
  - Use the partition principle, divide and conquer to partition the input space to make to process quicker and easier.
  - In this case, the requirement being verified will also mean it is validated.
  - This means there are only 4 tasks to be scheduled in the plan:
    - Generate orders data synthetically in multiple variations for a single validation condition
    - Build some scaffolding to simulate a system which scans and validates the orders
    - Partition testing data into 2 main categories: payment info and order info and test both groups together.
    - Design a logging system to record the outcome status of each order after validation compared to the expected status.

Scaffolding and Instrumentation

For requirements:

- **SR-1:**
  - A simulator of a drone to test the delivery capabilities. Because this is scaffolding, it must be placed in the early stages of development.
  - Having data to test the simulator. However, for it to work, the system requires components to parse the data correctly, built later. Therefore, some effort will be scheduled.
  - The system will require combining the simulator with the generated data to observe the results.
  - Additionally, designing and instrumentation to log the drone performance needed to be scheduled and also tools for analysis are required.

- **IR-2:**
  - Data generator which would generate data according to the data specification given to it.
  - Partition the input space. Grouping payment attributes and grouping other order attributes together results 2 partitions to be tested.
  - Using parsing and validator components to validate the data to get the output if the order is valid or not.
  - Design another logging system to record the outcome status of each order after the validation process which will be compared to the expected value

Process and Risk

Building the simulator for the first requirement can be done at an early stage of system design however generating data will not be possible at that stage as at least data parsing components are needed. Therefore, data generation will have to be done at a later stage of model design, but this can be done concurrently with the other development tasks. There can also arise an issue with the SR-1 requirement data. How representative is it? Is it realistic? What if the load was higher? These factors could significantly affect the result. Another issue we might have is a scheduling-related risk. To complete a task of orders validation, each validation criterion first needs to be tested separately. However, testing of the IR-2 requirement is done at the integration testing stage. Therefore, poorly unit testing of those components can produce errors at the integration testing stage. Furthermore, if the input space will be partitioned into 2 groups, then it will be even more difficult to analyse why some specific groups failed the test.