Stated test approach for select requirements to show range rather than every single one

Functional requirements:

- System must be able to read REST-Server data – Component/Unit
- System must be able to turn REST data into a usable format – component, integration
- System must validate orders (Robustness) – component
    - **Test approach:** Partition and use lots of small unit tests for the components with synthetic orders and then unit test for when they are combined in the final validation method.
    - **Evaluation:** This approach lends itself well to TDD and provides fine-grain testing for verifying the requirement, however a limitation is that it does not test if the validated orders can be handled by other modules.
- System must build no-fly-zones using data from REST server – component, integration
    - **Test approach:** Unit test no-fly-zone builder using designed polygon inputs in the form of polygon object instance, then integration test by combining the REST data retrieval component, feeding the resultant polygon objects as inputs into the zone builder.
    - **Evaluation:** This approach is good for testing the core functionality of the requirement. A limitation of this approach is that it does not account for incompatible formatting of REST data or the failure to access the REST server.
- System must only deliver the valid orders for the current day – integration, system
- System must calculate the optimal route for the drone – component, system
- System must route the drone around no-fly-zones – integration, system
- System must return the drone to the start point (Appleton Tower) at the end of a day's route - system
- System must facilitate bank payments - integration
- System must allow restaurants to register - component
- System must allow restaurants to create their menu - component
- System must select most expensive orders to deliver to maximise profit - system

Qualitative requirements:

- System must be reliable - system
    - **Test approach:** Stress testing the system multiple times by giving it a large dataset of unvalidated REST server orders to see if and how the system breaks.
    - **Evaluation:** This approach provides rate of failure data on the upper extreme of order load which is important for ensuring the system is reliable. However, it does not provide MTBF or POFOD during normal load conditions which are the most important metrics since the system only runs for a short amount of time each day.

Measurable requirements:

- System must keep the drone's move counter as low as possible for each delivery to maximise total deliveries – component.
- System must not give the drone a daily route more than 2000 moves long – component, system.

Performance requirement:

- System must complete the day's route calculations within 60 seconds – system timing performance.
  - **Test approach:** Create a timer to monitor the running of the algorithm. Create system stress tests to see the time taken to complete test cases of various number of orders.
  - **Evaluation:** The varying test cases of volume of orders is a good way to test the requirement and to indicate the point at which the fails to be met. However, a limitation is that restaurants are at varying distances from the start point so the make up of the orders is a factor that isn't being tested.

Security requirements:

- The software must keep the payment information of customers secure.
- The software must keep the drone's daily flight path secure to prevent interception.

Robustness requirements:

- If there is an error in calculating the flight path for an order in the list, the system should be able to move onto the next orders instead of getting stuck on the erroneous one. – System test
  - **Test approach:** System tests in which we provide a dataset including an order that will induce an error and observing/evaluating the chosen criteria for requirement verification.
  - **Evaluation:** This approach fits the requirement to a degree since it tests the ability to recover from an error. However, it is limited in that it is only testing for errors that we can identify AND induce in a controlled manner.