***ILP Planning document***

# **Requirements to be considered**

In this document, I am going to consider only 3 of the requirements from section 1 and analyze them in detail in order to create a test plan analysis. The requirements I am going to consider are:

* **Requirement 1:** *“The system should ensure that the drone never enters a no-fly-zone”*.
* **Requirement 2:** “*The system should validate each order for the given date, and, in case some of the order information is not valid, label it with the appropriate order outcome depending on which information was invalid”*.
* **Requirement 3:** *“The system should be able to retrieve data about different components (such as location of the University’s Central Area, locations of No-Fly-Zones, restaurant locations and order information) from a given REST server”*.

The selected requirements vary in their respective levels: requirement 1 is a **system** level requirement, requirement 2 is an **integration** level requirement, and requirement 3 is a **unit** level requirement.

# **Priority and pre-requisites**

In this section, we consider each of the 3 requirements, and provide a short assessment of its T&A (Test and Analysis) needs.

1. **Requirement 1:**

* This is a safety requirement with implications for several stakeholders, so it will be a high priority to ensure we meet this requirement
* Therefore, we will need to devote a high level of resource to ensure the requirement is met
* Since the requirement is a high priority, we will need to consider at least two different T&A approaches. These approaches are going to be ***partition*** and ***redundancy***
* Although this requirement is a system level requirement, checks can be performed in order to detect bugs earlier in the development process. This might increase T&A effort in the short term but would result in less checks being performed at the later stages of development.
* We will also need to consider the possible inputs and outputs of the path finding algorithm:
* **Inputs:**
* A list of orders for the given date
* A starting location on the map (for the first order, this will be Appleton Tower)
* The remaining number of moves that the drone has
* **Outputs:**
* Can\_be\_delivered: a Boolean (True/ False) value that indicates whether or not the drone has enough moves left to deliver the order
* If can\_be\_delivered is True, then also a list of points constituting the flightpath the drone takes in order to make the delivery
* **Specification:**
* An order is only delivered if can\_be\_delivered is True and every point in the flightpath is not inside a no-fly-zone
* The partition principle suggests that to help ensure safety we should decompose the requirement into a ***functional component*** that ensures a drone’s path to a point on the map doesn’t cross a no-fly-zone, and a ***structural component*** that ensures the drone never enters a no-fly-zone when delivering orders for a day. Therefore, this principle suggests we get two different aspects that will need to be considered in the plan:
* Some sort of inspection of the path planning algorithm to ensure the requirement is properly implemented by specific code and that there is no extraneous code that cannot be seen as implementing the requirement.
* A later exhaustive test that checks several combinations of order inputs and checks the result conforms to the specification.
* The redundancy principle suggests that redundant checks can increase the capabilities of catching specific faults early or more efficiently. This relates to the point made in the partition principle, where this requirement is split into the two aspects mentioned above. These tasks can be further broken down into different types of checks being performed, such as ***validation checks*** for both tasks, ***static checks*** as part of the first task, and ***assertion properties*** as part of both tasks.
* These two principles suggest the following tasks should be added to the test plan:
* Generate synthetic data to test the flight path algorithm
* Building some sort of scaffolding to simulate the software and run the tests on the synthetic data earlier on
* Design more extensive test cases at system level to ensure the software works as expected on real data (from the REST server)

1. **Requirement 2:**

* This is a correctness requirement at integration level, so it will have a medium priority
* Therefore, a moderate level of resource (greater than for requirement 3, but less than for requirement 1) will need to be devoted to ensure the requirement is met
* Since the requirement is of medium priority, we will consider one of the T&A approaches, namely ***redundancy***
* The redundancy principle suggests that redundant checks can increase the capabilities of catching specific faults early or more efficiently. In the case of this requirement, we can use various ***validation/ verification checks***, as well as ***static checks*** on synthetically generated data, in order to ensure this requirement is met.
* We will also need to consider the possible inputs and outputs as part of this requirement:
* **Inputs:**
* Current\_order: the order object being considered
* Outcome: the outcome field for the current\_order object
* **Outputs:**
* is\_valid: a Boolean (True/ False) value that indicates whether or not current\_order is a valid order
* the outcome field being set to one of the 10 possible values (***ValidButNotDelivered, Invalid, Delivered, InvalidTotal, InvalidPizzaCount, InvalidPizzaNotDefined, InvalidCvv, InvalidExpiryDate, InvalidCardNumber, InvalidPizzaCombinationMultipleSuppliers***) according to whether or not is\_valid is True or False
* **Specification:**
* is\_valid is only set to True if all the validation checks pass, and the outcome field is set to ***ValidButNotDelivered***; otherwise, if an order is invalid, the outcome field is set to the appropriate invalid label
* These points made so far suggest the following tasks should be added to the test plan:
* Generate synthetic data to test the flight path algorithm
* Building some sort of scaffolding to simulate the software and run the tests on the synthetic data earlier on
* Design more extensive test cases in order to integrate the data obtained dynamically for the REST server with the order validation

1. **Requirement 3:**

* This is a liveness requirement tested at unit level, so it will have quite a low priority
* There are validation and verification checks that will need to be performed.
* Information about the different data from the REST server is needed (characteristics such as length of data input, as well as specific information relating to individual pieces of data (e.g., order details))
* To validate and verify this requirement, a simple unit test that checks for correct functionality should suffice
* The testing for this requirement can be done at any stage of the development process (either early on, or at a late stage, depending on the time and resources available to spend on the requirement)
* The following inputs and outputs need to be considered as part of this requirement:
* **Inputs:**
* **Central Area:** the set of co-ordinates that constitute the University’s Central Campus Area
* **No-Fly-Zones:** the list of all no-fly-zones that the drone should avoid when making deliveries
* **Orders:** the list of all orders available on the REST server, for all possible dates
* **Orders for a given date:** the list of all orders for a single specific date
* **Restaurants:** the list of all the restaurants available on the PizzaDronz app
* **Outputs:**
* **Central Area:** there are 4 corners
* **No-Fly-Zones:** there are 4 no-fly-zones
* **Orders:** there are 7050 orders in total
* **Orders for a given date:** there are 47 orders for any given date
* **Restaurants:** there are 4 restaurants in total

# **Scaffolding and instrumentation**

In this section, I am going to describe what scaffolding and implementation are needed in order to carry out the given tasks (and this may give result in more tasks to build scaffolding and instrumentation). For the selected requirements, we have the following:

1. **Requirement 1:**

* Some sort of simulator for the system, so that it’s possible to test the software. This is scaffolding and will need to be scheduled early.
* Having data for the simulator may involve some effort that needs to be
* scheduled
* The system test will combine the simulator with the actual data obtained from the REST server
* The previous tests (before system level) will combine the simulator with the synthetically generated data

1. **Requirement 2:**

* Some sort of simulator for the system, so that it’s possible to test the software. This is scaffolding and will need to be scheduled early.
* Having data for the simulator may involve some effort that needs to be
* scheduled
* The integration test will combine the simulator with the actual data obtained from the REST server

1. **Requirement 3:**

* This section will not require any scaffolding since a simple unit test will be enough to verify this requirement.
* Therefore, the only instrumentation needing to be scheduled is the implementation of the unit test

# **Process and Risk**

**Requirement 1:** *“The system should ensure that the drone never enters a no-fly-zone”*.

This requirement could take several days, possibly even more than a week, to complete, depending on the software complexity. It would require access to data on restaurant locations and the no-fly zone borders, as well as the details about those no-fly zones (for example, whether they can exist outside of the central area, etc.). This data can be obtained from the REST server, which falls under requirement 3, and will therefore also be subject to the limitations mentioned there. This task can be placed in the elicitation and analysis stage of the SRET lifecycle process, as it involves identifying and addressing safety concerns. It could also be included in the design and implementation stage, since it specifies an important constraint the software must comply with.

The main risk for this requirement is that the navigation design process might be highly complex and/ or take up too much time/ resources, causing delays and inefficiencies. Additionally, the software might malfunction, leading to unintended or unsafe drone moves, potentially entering some no-fly-zones. Therefore, we need to consider streamlining the process and identifying opportunities for automation and/ or optimization, whilst also ensuring the software is tested extensively to identify and avoid any possible issues.

**Requirement 2:** “*The system should validate each order for the given date, and, in case some of the order information is not valid, label it with the appropriate order outcome depending on which information was invalid”*.

This task could take around a day or two to complete. Access to the REST server would be needed, and the server should already contain at least some sample orders. In addition, precise information on validation details is also a must. This requirement could be placed in the design and implementation stage of the SRET lifecycle process, since it specifies the validation process the software should follow.

The risks for this requirement mainly relate to issues with the REST server. For instance, the information there could be incomplete. This could potentially lead to invalid orders being accepted or rejected, depending on the validation criteria being used. Therefore, it is important to ensure additional checks or verification steps to check the data on the REST server is complete. Moreover, it is possible that the REST server fails or becomes unavailable, so it is important to have alternative sources or contingency plans. On the other hand, it is also possible that the validation process could be highly complex and/ or take up too much time and/ or resources, causing delays and inefficiencies.

**Requirement 3:** *“The system should be able to retrieve data about different components (such as location of the University’s Central Area, locations of No-Fly-Zones, restaurant locations and order information) from a given REST server”*.

Like in requirement 2, it is possible that the REST server might fail or become unavailable, so it is important to make sure to have alternative sources or contingency plans. Another risk, which again is similar to requirement 2, is the information on the REST server could be incomplete, and this could have quite a serious impact on the viability of our system, especially if we consider restaurants (where the orders are meant to be delivered from) and no-fly-zones (the areas meant to be avoided). Therefore, for this requirement, it is critical to ensure additional checks or verification steps to check the data on the REST server is complete.

## **Evaluation of instrumentation**

The current instrumentation provides quite thorough testing through combining the synthetically generated data used at unit level with automated integration testing using the real data. In addition, the planned automated tests should guarantee enough coverage to give us the confidence that the system works as expected, or at least allow us to find and fix all necessary bugs.

However, one way to further improve the robustness of our system would be through manual testing. This would also resemble acceptance testing, since it will mimic an actual user trying to use the program, rather than the process being automated. However, one disadvantage to this approach is that it will most likely be the developer performing the manual testing; this is a drawback since the developer knows the underlying mechanisms for the system and might not try all possible combinations that an inexperienced user might attempt. In addition, this will require additional time and resources, which could be problematic given the tight schedule.

Another drawback is that the data retrieval is only tested with the actual REST server. This could be improved by having another server, a synthetic REST server, with synthetic data that could be used for testing. This could also reduce the need to create new synthetic data manually. However, the drawbacks would again be the additional time and resources needed for performing this task.