LO1: Requirement Analysis for PizzaDronz Project

1. Range of Requirements

1.1. Functional Requirements (*S – system, I* – integration, *U* – unit level):

R1 *S* • The application shall accept exactly 3 arguments: the date when the orders are to be delivered, the base URL address of a REST server and any word.

S • The application shall retrieve the following data from the REST server:

U ➢ A list of participating restaurants.

U ➢ The central area.

U ➢ No-fly zones.

U ➢ A list of orders on a given date.

R2 *S* • The application shall validate the orders based on the following criteria:

U ➢ The order number must be an 8-digit hexadecimal number.

U ➢ Credit card number must be a valid VISA or MasterCard number.

U ➢ Credit card expiry date must be after the order date.

U ➢ The CVV must be valid for the card types mentioned above.

U ➢ The order must contain between 1 and 4 pizzas inclusively.

U ➢ All pizzas in the order must match the pizza names on the restaurants' menus.

U ➢ All pizzas must be from the same restaurant:

U ➢ The total cost of the order must be equal to the sum of pizza prices plus the fixed delivery fee of £1.

S • The application shall compute a valid flightpath of the drone for the entire given day so that some/all orders are delivered. A flightpath shall move-by-move specify how the drone moves while delivering orders on that day. The validity of a flightpath shall be defined by the following criteria:

I ➢ Drone's moves must be defined by the drone's characteristics:

U - The drone can make a fixed maximum number of moves: 2000.

U - Each day the drone starts fully-charged, and once the battery is exhausted it does not operate on that day anymore.

U - Each move is either a straight line of 0.00015 degrees or a hover move.

U - The drone can only fly in the 16 defined directions: E, ENE, NE, NNE, N, NNW, NW, WNW, W, WSW, SW, SSW, S, SSE, SE, and ESE. Here E is East, N is North, W is West, and S is South.

U ➢ The drone must on any given date start deliveries from the top of Appleton Tower (AT) and return *close to* it when all orders are delivered or it cannot deliver any more orders due to the battery being exhausted.
Close to a certain destination means that the drone is within a fixed distance of 0.00015 degrees from that destination.

U ➢ The drone must hover for one move when collecting an order from a restaurant and when delivering it on top of Appleton Tower (AT).

U ➢ When hovering for collection/delivery above a restaurant/AT, the drone must be *close to* them.

I ➢ When returning from a restaurant to AT, the drone must not leave the central area once it has entered it. [1]

I ➢ The drone must not at any point in time be inside a no-fly zone, nor must it cross any borders of a no-fly zone. [1]

R3 *S* • The application shall associate each order on a given date with a certain outcome. The possible outcomes shall include *delivered*, *valid but not delivered* and *invalid* with a reason for invalidity.

S • The application shall create 3 output files:

U ➢ A file which contains JSON records corresponding to each order on a given date. Each record must contain the order number, the outcome associated with the order and the total cost of the order.

U ➢ A file which contains JSON records representing every single move the drone makes on a given date. Each record must contain the order number currently being delivered, location of the drone before the move and after it (in terms of longitude and latitude), angle of the move and the value in nanoseconds representing the time elapsed since the start of computation.

U ➢ A file which contains a list of coordinates in GeoJSON format which illustrates the flightpath of the drone.

1.2. Non-Functional Requirements:

- Performance (measurable attribute):
  - ➢ The application should run for less than 60 seconds before termination.
- Efficiency:
  - ➢ The average number of delivered orders per day should be maximised as an efficiency metric. Thus the application should be able to find an optimal flightpath for the drone.

R4

- Robustness:
  - ➢ In case of any incorrectness in the arguments supplied to the application, the application should terminate and inform the user about the error.
  - ➢ In case of any missing data on the server side, the application should continue if the missing data can be omitted in the calculations, and terminate and inform the user of the cause otherwise.
  - ➢ In case of any other errors occurring during the execution of the application, they should be handled by terminating the application and informing the user about the cause of the error.
- Extensibility:
  - ➢ The application should be designed in such a way that it can be relatively easily extended.
- Security (a potential non-functional requirement):
  - ➢ All users' personal data should be stored securely, with the chance of personal data theft due to an attack minimised.

[1] These two requirements are marked as integration requirements because they comprise all requirements for the drone's move correctness plus the specific restrictions (central area and no-fly zones). Thus they cannot be tested on the unit level.

2. Level of Requirements

All functional requirements in the previous section are marked with a certain symbol – *S, I, or U* – for system, integration and unit level requirements respectively.

The most general are the system-level requirements, i.e. what functionality the application shall provide. The system-level requirements often have certain criteria, based on which they should be satisfied. These criteria are usually considered as the unit-level, sometimes as integration-level requirements. This is due the nature of those criteria: if they specify a single, concrete functional requirement, they are treated as unit-level; if they comprise several unit-level requirements, they are treated as integration-level.

All non-functional requirements are obviously system-level requirements as they are only applied to a completed application.

3. Testing Approach for Chosen Attributes

Unit testing, integration testing and system-level testing are to be used for the three corresponding levels of requirements. Besides that, some statistical testing should be used to find the sampled average program execution time (performance) and the sampled average number of delivered orders per day (efficiency).

4. Assessment of the Appropriateness of the chosen Testing Approaches

The tests are well-matched to the types and levels of the requirements. They will guarantee the functional requirements are met, and potentially some non-functional requirements (particularly, performance and efficiency) are satisfied. Moreover, due to the fact that the system-level requirements usually depend on the smaller unit-level requirements, the tests would be developed accordingly and ensure certain correctness features are tested on their own (unit), as well as together with other correctness features that comprise the same system-level requirement (integration, system).

One potential deficiency is in the choice of unit testing of data retrieval from REST server. The unit tests would ensure the retrieved data is what is actually expected, but would not guarantee a correct behaviour in case of some missing data on the server side (robustness problem). Also, since some qualitative requirements are not measurable, such as in our case security and extensibility, they cannot be guaranteed by the chosen testing approaches.