Software Testing 2024/2025 Portfolio - Requirements

The requirements discussed in this document are based on the Informatics Large Practical specification for the PizzaDronz application.

Requirement list:

First we can discuss the requirements that concern the drone movement and pathing part of the system:

1. **Drone should deliver pizzas to Appleton Tower:** *Functional, Quantitative*
   This requirement specifies what the basic pathing system must do and provides a final, measurable goal (that the drop-off point is Appleton Tower).
   Requires Unit testing.

2. **Drone must start from Appleton Tower:** *Functional, Quantitative*
   As with requirement 1, this further specifies what the system must do and provides the measurable outcome of the drone starting its journey from a specific point (Appleton Tower).
   Requires Unit testing.

3. **Drone must return to central campus area after collecting the pizza and can not leave until pizza is delivered:** *Functional, Quantitative*
   The expected functionality of the drone is made further specified in this requirement. It is something that can be tested and measured to ensure the system adheres. ***Requires Unit testing.***

4. **Drone hovers for a suitable amount of time on both delivery and collection:** *Functional, Quantitative/Qualitative*
   This requirement describes the behaviour of the drone at drop-off and pick-up points. As we are able to describe a 'hover' through a movement with direction 999^o at these points, we have a measurable requirement. However, a 'suitable amount of time' is subjective to the customers and participating shops and this aspect of the requirement would require further evaluation to ensure it is met.
   Requires Unit testing.
   Requires further non-automated evaluation.

5. **Drone moves in the 16 major compass directions:** *Functional, Quantitative*
   This requirement simply defines the directions in which the drones will be able to move. It is a foundation of the software for the drone movement and is a measurable in testing it is followed.
   Requires Unit testing.

6. **Drone should avoid popular areas:** *Functional/Non-Functional, Quantitative*
   This requirement details an important function of the system. The reason I have labelled it as both function and non-functional is due to the fact that 'avoiding popular areas' is a concern of safety and privacy (so not directly linked with the functionality), however, if these areas are specifically given to the drone, it is expected the drone will avoid these (this is a function of the system that can be measured).
   Requires Unit testing.
   Requires further non-automated evaluation.

7. **Drone should fly over rooftops where possible:** *Non-Functional, Qualitative*
   As above, this requirement is concerned with the safety and privacy of all stakeholders in this application. This specific requirement however is not readily measurable as the drone itself will be unable to determine if it is over a roof and this data is not provided.
   Requires further non-automated evaluation.

We also must discuss the requirements of order and payment validation. This ensures that the correct orders should be, and will be, delivered:

8. **Orders must be validated:** *Functional, Quantitative*
   A fundamental requirement for the system which can be, and must be, tested vigorously.
   Requires Unit testing.

9. **An order can only contain 1-4 pizzas:** *Functional, Quantitative*
   This requirement defines an operational constraint with measurable limits.
   Requires Unit testing.

10. **Price of order must be the price of all pizzas in the order plus an ordering fee:** *Functional, Quantitative*
    This requirement ensures further correctness in the validation and is something that can be measured and tested.
    Requires Unit testing.

11. **All pizzas in one order must come from the same restaurant:** *Functional, Quantitative*
    As above (requirement 10).
    Requires Unit testing.

12. **The restaurant being ordered from must be open:** *Functional, Quantitative*
    As above (requirement 10).
    Requires Unit testing.

13. **Credit card numbers must be 16 digits long, with no characters:** *Functional, Quantitative*
    As above (requirement 10). There is also the need for valid card details on the security front and so this requirement will help for this to be considered.
    Requires Unit testing.

14. **Credit card expiry dates must be in the future, and must be actual dates in the format of MM/YY:** *Functional, Quantitative*
    As above (requirement 13).
    Requires Unit testing.

15. **Credit card CVVs must be 3 digits:** *Functional, Quantitative*
    As above (requirement 13).
    Requires Unit testing.

And finally we look at the general system requirements that are not covered by the order validation or movement handling aspects of the system:

1. **The system must run when provided with correct input:** *Functional, Quantitative*
   This may be the most obvious and basic requirement but having a running system for appropriate input can and will be tested and measured.
   Requires System testing.

2. **Runtime is 60 seconds or less:** *Non-Functional, Quantitative*
   An important requirement is the speed at which the system fetches and processes all orders for a day. Having a strict limit is measurable and provides a tangible goal.
   Requires System testing.

3. **Upon failure, the system should display an appropriate error message:** *Functional, Qualitative*
   This requirement ensures those running and using the application can see what went wrong and what needs to be changed/fixed with either their inputs (i.e. dates or URLs) or maybe even the code itself.
   Requires Unit testing.
   Requires Integration testing.

4. **The system must connect to and correctly fetch data from the REST server:** *Functional, Quantitative*
   This requirement specifies a clear action of fetching all required data (e.g. order data, restaurant data etc.) with a specific and measurable goal of correctly fetching and parsing the data provided on the REST server.
   Requires Unit testing.
   Requires Integration testing.

Requirement variety:

The range of requirements and the variety in their levels are both clearly shown (in the list above it should be noted there is diversity in functional and non-functional requirements, quantitative and qualitative requirements, and requirements requiring unit, integration or system testing). It is important to ensure an array of different requirements are discussed and met in the system as to solidify the success of the project.

Test approach:

Identifying the test approach:

As mentioned at the start of this document, the requirements are developed from the course specification for Informatics Large Practical. It therefore makes the most sense to have a focus on a specification-based testing design looking heavily at the functional accuracy of the program, along with error detection and compliance. Testing this system will therefore mostly use requirement-based testing, equivalence class partitioning and random testing.

Assessing the appropriateness of the test approach:

My test approach prioritises a specification-based testing design and will effectively build up tests from the project specification itself as it will ensure the system and its functionality directly corresponds with the detailed requirements given by said project specification. It makes sure that each functional requirement of the system is met and thoroughly tested. However, this may means that my testing approach will avoid some other designs (such as structural and experience-based testing). These other testing designs would provide further feedback and help to maximise the success of the project, however the small scale of the project along with the lack of time and resources make it not feasible to combine my test approach with these other designs. All in all, my current approach is not the most optimal, but with the resources available to me, it is the best approach in terms of ensuring functional accuracy.