Test Planning

This test plan is meant to represent an initial testing plan for PizzaDronz project based on the requirements (functional and non-functional) specified in the LO1-Requirements.pdf file.

For this project, the Agile software lifecycle model is best. Its primary benefit is that testing additional portions can begin immediately, without having to wait for a predetermined set of requirements to be fulfilled. As a result, the entire development process is effective for the test-driven design and very dynamic. We cannot possibly handle every need during the testing process due to our limited resources. Thus, we will generally talk about when and in what capacity testing for the satisfaction of each functional requirement as well as the performance, efficiency, and robustness non-functional criteria will occur. We will go into greater detail about a few criteria, including addressing the scaffolding and instrumentation requirements.

Procedure will be:

1. Unit testing during the development phase.

2. Integration testing following the completion of a particular capability and the successful completion of unit testing for its constituent parts (e.g., requirements for the center area and no-fly zones).

3. System testing following the completion of the entire application.

The application should be corrected, and the tests should be rerun if at any stage they fail specific level tests. Because test-driven design is used in the agile lifecycle, this process should be performed as often as needed until the tests pass and a requirement is met.

Functional Requirements:

Since they outline the fundamental features of the program, the functional system-level needs listed in requirements.pdf will be given priority. As a result, a significant amount of resources have to be set aside to satisfy them. This also relates to the unit and integration requirements at a lower level, which serve as the basis for some system-level needs.

It is imperative to incorporate V&V activities into the development process as early as feasible, since this will minimize the amount of time required for A&T due to early issue discovery. Our selected test-driven design will prioritize this, thus in order for the software to be deemed right, it must pass the pre-engineered tests. This idea mostly relates to unit tests since they can be completely designed beforehand, but it also affects integration and system tests, which can be created or modified after the unit tests have shown that the criteria at the unit level have been met. This method works well with the Agile lifecycle since it eliminates the need for a full test set to be created before any development begins.

Non-functional Requirements:

Since performance and efficiency are not related to drone movement regulations or other mandatory application features, non-functional requirements will be prioritized less. As a result, less resources ought to be allotted to satisfy them.

High priorty will given to robustness since it ensures that the program will function properly in the event of unexpected problems. By resembling mistakes, it can be fulfilled early on.

Early detection of unacceptable outcomes is not achievable since V&V operations for other non-functional requirements will only occur after the system is completed.

In order to go in detail with test planning I will be choosing following requirements:

Requirement1) System shall generate a flight path for a singular drone to deliver a pizza order, avoiding no fly zones and considering central area requirements.

Requirement 2) The system shall process and validate each order fetched from REST server to make sure its correctness and completeness. Details such as order number, order status, and the order validation code should be extracted from the REST server.

Requirement3) The system should consider two locations that have a Euclidean distance of 0.00015 between to be 'close' (so if the drone is 'close' to a point, we consider the drone to be at the point). This requirement could be categorized under functionality.

Requirement1) System shall generate a flight path for a singular drone to deliver a pizza order, avoiding no fly zones and considering central area requirements.

- Ensuring we achieve this criteria will be of the high priority because it concerns to safety and has impacts for multiple stakeholders.

Due to the high importance of this demand, we will need to take into account at least two distinct T&A methodologies. As a result, we will need to allocate a significant amount of resources to guarantee that it is met. These approaches are going to be partition and redundancy.

-Even though this need is at the system level, tests can be run to find flaws early in the development cycle. Short-term T&A effort might rise as a result, but later in the development process, less inspections would be made.

- The path finding algorithm's potential inputs and outputs must also be taken into account:

Inputs:

-A list of orders for the given date

-A starting location on the map (for the first order, this will be Appleton Tower).

-A list of no-fly zones

Outputs:

If asserFalse pathGoesThroughNoFlyZone is true then drone never enters no-fly zone

If asserFalse pathGoesThroughNoFlyZone is false then drone enters no-fly zone

According to the partition principle, we should break down the requirement into two parts: a structural part that makes sure the drone never enters a no-fly zone while delivering orders for the day, and a functional part that makes sure a drone's path to a point on the map doesn't cross one. As a result, this concept indicates that the plan should take into account two distinct factors:

- A path planning algorithm inspection of some kind to make sure that the requirement is appropriately implemented by the designated code and that no unnecessary code exists that would obscure the implementation of the requirement.

- A further comprehensive test that verifies that the outcome complies with the specification by examining multiple ordering combinations of the inputs.

According to the redundancy concept, doing redundant checks can improve the ability to identify certain errors and fix them faster or earlier. This connects to the issue raised in the partition principle, which divides this prerequisite into the two aforementioned elements. These jobs can be further decomposed into several types of checks that are being carried out, like assertion properties in both tasks, validation checks for both tasks, and static checks as part of the first task.

The following tasks need to be included in the test plan, according to these two principles:

-Synthetic data should be generated to test the flight path algorithm.

-A scaffolding of some kind should be built to simulate the software and run the tests on it earlier.

Scaffolding and Instrumentations:

-A simulator of some kind for the system, enabling software testing. Since this involves scaffolding, an early scheduling is required.

-Obtaining data for the simulator could need some planning and effort.

-The simulator and real data from the REST server will be combined in the system test; the simulator and artificially generated data will be combined in the tests conducted prior to the system level.

Process and Risk:

Process:

First priority should be given to developing the data injection method and path modeling tool.
Early on, the algorithm's basic functioning was tested using generated data. The simulator and algorithm are integrated, and compatibility and fundamental performance are tested.

Risk and Mitigation:

The primary risk associated with this requirement is that the navigation design process could be extremely time-consuming and/or complex, leading to delays and inefficiencies. Furthermore, there's a chance that the software will go awry and cause the drone to fly into prohibited areas or make harmful or unwanted movements. Thus, we must think about simplifying the procedure and finding chances for automation and/or optimization. We also need to make sure that the program is thoroughly tested to find and prevent any potential problems.

Mock data may not be representative and may overlook certain situations that could lead to the application's eventual denial.

To ensure that drones do not fly in any way that could be construed as illegal, the REST server must be updated about no-fly zones and the core area.

Requirement 2) The system shall process and validate each order fetched from REST server to make sure its correctness and completeness. Details such as order number, order status, and the order validation code should be extracted from the REST server.

-High priority because it is essential to PizzaDronz's basic operation.

-Important for guaranteeing accurate and comprehensive order processing, which has a direct bearing on financial transactions and customer satisfaction.

-Make certain that PizzaDronz can retrieve orders from the REST server with consistency and dependability.

- Develop and set up a system that reliably extracts order information from the REST server, such as the order number, order status, validation code, etc.

- Establish a regulated environment for testing that includes order data variances and mirrors the production system's characteristics.

- Early stakeholder interaction to ascertain expected format and order data differences.

Inputs:

-Order object being considered

-Restaurant object being considered

Outputs:

- OrderValidationCode set to one of these:CARD_NUMBER_INVALID, EXPIRY_DATE_INVALID, CVV_INVALID, TOTAL_INCORRECT, PIZZA_NOT_DEFINED, MAX_PIZZA_COUNT_EXCEEDED, PIZZA_FROM_MULTIPLE_RESTAURANTS, RESTAURANT_CLOSED,

-OrdeStatus set to one of these: INVALID, VALID_BUT_NOT_DELIVERED;

Scaffolding and Instrumentations:

- Mock REST server that simulates order scenarios with different order status and validation codes.

- Provide a way to simulate different order types and insert test data into the order processing system

- Logging mechanism to point out updates in order validation and status

- Error tracking and logging to point out problems that happened during order process

Process and Risk:

Process:

To verify individual functionality, the first stages of development concentrate on unit testing order processing components with mock data. This is done to provide a controlled environment in which these components can interact while also developing a data injection mechanism and a mock REST service. Integration testing, which uses a mock REST server to replicate controlled scenarios and verify how various system components interact, becomes increasingly important as the project moves forward. In order to monitor system activity and spot possible problems, instrumentation for logging and error tracking is deployed concurrently and is constantly improved. The logging mechanisms are further refined as a result of rigorous system-level testing carried out with a variety of data circumstances as the system evolves.

Risks and Mitigation:

-Mock data may not be representative and may overlook certain situations that could lead to the application's eventual denial.

- the REST server could have incomplete information, which could result in incorrect categorisation of orders.

- Update the mock REST server often to make sure it accurately depicts order variations in the actual world.

- It's critical to have backup sources or plans in place in case the REST server malfunctions or is unavailable, as well as checks in place to guarantee the data on the server is accurate.

Requirement 3: System shall not allow any point which is further away than the distance tolerance of 0.00015 degrees to be considered close. This is a unit level requirment, which should have a low priority in the development process.

-A mathematical formula that determines the Euclidean distance needs to be established and incorporated into the system.

-It is important to carry out validation and verification procedures to guarantee that the Euclidean distance is calculated accurately.

-To validate and verify the requirement, a straightforward unit test that verifies the calculation's right functioning should be put into place.

-Depending on the time and resources available, testing for this criterion can be conducted at any point during the development process.

-As part of this requirement, the following inputs and outputs must be taken into account:

<u>Inputs</u>:

Two locations with their coordinates (x1, y1) and (x2, y2)

<u>Outputs</u>:

Boolean value indicating whether the two locations are 'close' or not.

Scaffolding and Instrumentations:

Since this is only a unit test, there would be no scaffolding for this requirement.

Process and Risk:

There aren't many dangers involved in calculating the distance between two points because an error would only happen if the points were entered incorrectly or if the formula being used to compute the distance was incorrect.

EVALUATION

A key advantage is constant improvement, which guarantees flexibility in response to changing project requirements. The early development of data injection techniques and simulators provide a controlled testing environment. To avoid such bottlenecks, however, careful control of the documentation overhead is necessary. A significant dependence on synthetic data is introduced, and its representativeness must be carefully validated. Additionally, as the REST server itself ought to supply a sizable sample of data for testing, there can be more efficient uses of the available resources. In the end, I think it will be quite challenging to construct the mock REST server with simulation components, and as a result, it might not be finished.

Manual testing would be one more option to increase our system's robustness. Since it will simulate an actual user attempting to utilize the program rather than an automated procedure, this would also resemble acceptability testing. The manual testing will very certainly be done by the developer, which is a disadvantage as the developer is familiar with the system's core mechanisms and may not attempt all combinations that an inexperienced user may try. Furthermore, considering the short notice, this will take more time and resources, which could be an issue.