Scope:
Let us consider 3 requirements, which are as follows:

- R1 - The system should be able to find the shortest path avoiding no-fly zones.
- R2 – The system should take at most 60 seconds to find the shortest paths for all orders in each day.
- R3 - The system should be able to classify orders based on their attributes.

Out of scope:
As this is a student project with very limited resources and no key features (security system, transaction system, authentication system, actual drone model), I will not be able to plan and test some requirements, some of which are:

- The system should be secure and keep the users' data private and encrypted.
- The drone should fly over buildings.
- The restaurant owners should be paid what they earned in profit minus the service charge for using the system.
- The customers should be able to sign up, login, and place pizza orders consisting of a maximum of four pizzas from the same restaurant.

Assumptions/Managing expectations:
In this test plan, we will assume that there are multiple people working on this project such that each chosen lifecycle can be carried out as they should be. Currently, this project is only being developed by one person, thus in actuality, I will be carrying out the testing of the system after the code implementation stage following the V model development process, since I had already mostly finished my coursework prior to writing this test plan. Furthermore, there will be certain risks that could certainly arise if this was being developed by a company, however since I am the only developer on this project, and this is only a simulation of a drone project, there are no risks such as personnel risk or technology risk.

Level of resources dedicated:
R1 – High

- Viability of the system - many stakeholders such as the customers and restaurant owners depend on the system working as intended so that they can receive their pizza orders or make profit.
- Safety and privacy – the drone could cause physical harm, noise disturbances or record citizens without their consent if it enters a no-fly zone since it could be equipped with a camera, regulatory software requirements

R2 – Medium

- The priority level depends on the severity of faultiness that the pathfinding algorithm poses. If it manages to find a sufficiently short path, then spending additional resources into optimising it might not give back the same amount of saved time, and those developmental resources will be better utilised somewhere else.
- However, if the algorithm struggles to find a path in reasonable time or even one at all, then of course this will become a critical task, as an algorithmic fault in this case will mean that there has not been sufficient unit testing been done. (PathFinder and Drone class in this case).

R3 – Medium

- It depends on whether the system can correctly classify valid orders but attach wrong order outcome values to invalid ones. In that case, this would be a medium level priority requirement since as long as valid orders are marked to be delivered, it does not matter what the invalid orders are classified as.

- However, if invalid orders are marked as valid and vice versa, then this becomes a high-level requirement as it will be a detriment to all major stakeholders (customers, restaurant owners, me) if the system does not perform as expected.
- Therefore, to make sure that this system is functional, I believe that a medium-high level of resources should be dedicated to validating and verifying this requirement.

Chosen lifecycles:
R1 – SRET (Software Reliability Engineer Testing)
R2 – DevOps
R3 – XP (Extreme Programming)

Explanation/Testing strategy:
R1:

- I have chosen the SRET lifecycle since this is a system level requirement which I have to incrementally build from the ground up. Starting from unit testing the different classes such as LngLat (deals with coordinate calculation), Polygon (deals with checking if a point is in a polygon) and a custom-built Graph class.
- Then after unit testing, we need to make sure that the different components (pathfinding algorithm, drone movement algorithm and order validation component) not only function correctly by themselves but also when they are integrated, thus integration testing is also necessary prior to the system level test.
- Since this is a relatively small system compared to other big projects, we employ a structural strategy, more specifically the bottom-up strategy to carry out integration testing.

R2:

- DevOps seemed to be the most fitting lifecycle for R2, as not only can this be tested prior to the launch of the software, but it could be that the shortest path to the restaurant does not actually yield the shortest time to travel, thus recording and integrating real-life data into the development cycle will be imperative to continuously improve the algorithm. This will mean that the algorithm's runtime could change each iteration, and performance testing to keep satisfying this requirement is necessary will also be included within the DevOps cycle.

R3:

- I have chosen the Extreme Programming lifecycle for this, as this requirement only needs to test two separate classes – an Order class which deals with validating the orders, and a Card class which verifies that a card's details are all correct. As the model classes (like Restaurant, OrderOutcome, etc) would be completed already prior to coding the Order and Card class, testing this requirement would not take long.

Scaffolding and instrumentation:
R1:

- One aspect of this requirements would benefit from having some scaffolding, namely the pathfinding algorithm. The scaffolding could comprise of a random coordinate generator, which would produce several LngLat points to make up the different restaurants.
- Another piece of scaffolding necessary is the Java JGraphT library, which will be used to confirm that my custom graph implementation works and performs correctly.

R2:

- For this requirement, we could have a logging system within the software as a piece of scaffolding to record how long each part of the algorithm takes and identify problematic parts which could be improved.

R3:

- A piece of scaffolding that will used to test this requirement is dummy card values such as different card number values that do or do not pass Luhn's Algorithm, as well as other made-up card details and order information.
- Another instrumentation that I could use to devise the tests is a pairwise combination generator https://pairwise.yuuniworks.com/ which will allow me to input different attributes that a Card class could take and output different test cases by varying two values.

Evaluation of the scaffolding and instrumentation:
For R1, the input spaces that the driver generates could be too general. For example, it might output a set of points which are not a feasible area in a real-life scenario. To combat this, I could limit the range of coordinates that the driver generates. Furthermore, I will be re-using the no-fly zones provided in the Informatics Large Practical coursework, which will be an adequate real-life example of such areas. However, this will limit the types of no-fly zones that will be used for testing, and thus there could be polygon shapes which break my system. One example of how this could occur is outlined in LO2.2. Using the JGraphT library, this will ensure that my algorithm is functioning correctly as it is an official Java library which other users also employ, and I can check that the outputs of the Dijkstra's and A* algorithms match that of mine.

Regarding R2, the logging instrumentation system could be more detailed, and it could log additional information about how much space the algorithm is taking up while calculating the shortest path, and I could optimise not only the runtime but the space complexity as well.

With R3, although the made-up card details seem to be sufficient in covering most if not all scenarios, there will be certain edge cases that could be missed. However, given that these details are industry standard for testing different card types, I believe that this will be sufficient enough to ensure that the my system will at least accept Visa, Mastercard, and AmEx cards.

Risks and vulnerabilities:
R1:
- Scheduling risk
  - Due to the need for unit testing, integration testing and finally system testing, in a practical scenario, this would need to be thoroughly planned out and extra time allocated between the end of one task and the start of another to make sure that even if a delay occurs, additional tasks are not further pushed back.
- Development risk
  - Because this requirement sees many classes integrating together, there is the risk of each unit class or integration not being tested thoroughly enough. To mitigate this in the real-world, we could require some sort of criteria like coverage to be met at the unit or integration level. Furthermore, there could be inspections of unit tests done by senior engineers such that errors are caught before they are integrated.

R2:
- Personnel risk
  - If there is only a small number of engineers dedicated to keeping the system operational, it means that there is a risk of these engineers leaving the team and thus the company will lose key personnel and knowledge on how the custom algorithm functions and what ways there are to help the runtime be more efficient.

R3:
- There is no real risk for this requirement – it only deals with two small classes, and the code is intuitive enough that anyone could look and understand what it is trying to do, and therefore test it effectively. Furthermore, there are no additional third-party technologies

being used, so none of the engineers need to be trained specifically for achieving this requirement.

Evaluation:

All parts for each of the requirements are to be tested, with emphasis on achieving the first system level requirement as it involves many moving pieces in terms of developing the classes and the unit, integration, and system level tests. A bottom-up strategy will mean that each part of the requirement will be thoroughly tested before moving on, which will aim to ensure that the only errors present will only be the ones that come from the code in the class being tested, and not the ones from before. All vulnerabilities have been identified for each requirement, with a focus on development risk for R1 and personnel risk for R2 while there aren't any for R3 as it deals with a straightforward requirement that does not need extensive scaffolding to test it.