1.1

functional requirements

Drone should not fly over buildings in the no-fly zone.
Software should be able to access the database to get the information of orders.
The drone must at all times remain strictly inside the confinement area.

measurable quality attributes

The drone should deliver at least most of the orders.
The length of the path of the drone should not be longer than 1500 moves.
The application to plan and plot the flightpath of the drone should aim to have a runtime of 60 seconds or less.

qualitative requirements

Software should be data-driven.

it must read the information from the database and the website and particular shops or particular drop-off points or other details must not be hardcoded in the application,

it is being created with the intention of passing it on to a team of software developers and student volunteers in the School of Informatics who will maintain and develop it in the months and years ahead when the drone lunch delivery is operational. For this reason, the clarity and readability of the code is important; I need to produce code which can be read and understood by others.

1.2

unit requirements:

define database table for orders
define database table for flightpath
collect information about server and ports.
get angle of the current point to another point
get the sequence of areas which the drone will pass by
get the area number of the location of the drone
get the center of the current area
get the information of buildings from the web server
get the information of orders
get three word address
get the details of the points
get menu information
calculate total cost of the order
obtain location of shops according to the products
obtain order number and items
define datatype for three word address
get target orders

integration requirements:

consistent interpretation of parameters or values
match modules properly
avoid memory leak

system requirements:

retrieving information of orders from database
designing flightpath
recording the path in database

1.3

attribute1

It is accepted that the service may not be able to deliver every order every day depending on the number of orders placed. An important metric to be used in determining the viability of the drone delivery service will be the sampled average percentage monetary value delivered by the service.
The percentage monetary value delivered each day is calculated as the total monetary value of deliveries made divided by the total monetary value of orders placed. The fixed delivery charge of 50p per order is included in both totals.
The sampled average percentage monetary value is calculated by taking a random sample of days (say 7, 12, 24, or 31 days) and computing the average of the percentage monetary value delivered on each of those days.

attribute2

Everytime when there is a movement or howering, add one to movement index and output the result to check if it meets the requirement.

attribute3

use time() function to get the runtime

1,4

approach1

limitation: A small sample of days will only give an approximate idea of the viability of the service; larger sample sizes will give a more accurate idea which costs a lot.

approach2

it can work properly.
limitation: make the code more complex which increases the difficulty of readability.

approach3

it can work properly.
limitation: make the code more complex which increases the difficulty of readability.

R1： Drone should not fly over buildings in the no-fly zones
R2： The application to plan and plot the flightpath of the drone should aim to have a runtime of 60 seconds or less.

2.1

R1：
Some of the schools do not want to deal with dangerous problems caused by the drones so no-fly zones would be defined.
This is a safety requirement which may be related to regulatory requirements so high level of resource would be used in order to meet the requirement.
At least two different T&A approaches would be necessary.
This requirement can only be tested after the development of the software so early approaches cannot be used.
Redundant checking would be applied to find faults.

The path would be displayed after designing the path so we can use the inspection to check the performance of the software.

Use scaffolding to do exhaustive test is also possible.
Details of orders for a whole year is provided in the database.
Use scaffolding to run the software for everyday and check if the drone moves in the no-fly zones automately.

The testing software can be done in the early stage when modules are developing because the specification is not complex.

R2：
The priority of this requirement is not as high as the first requirement so it would not be allocated with much resource on it.
This is a measurable attribute of the code.

This requirement can only be tested after the development of the software so early approaches cannot be used.

There are validation and verification issues with this.
Synthetic data and logging of performance of the system are required.

Use scaffolding to do exhaustive test and use function to get the peocess time for each run and record them.
The function for producing synthetic data can be developed in the early stage. The requirement can only be tested after the completion of the software.

2.2

R1:

The first technique can check the flightpath easily but it costs huge human resources. Human also makes mistakes so this technique cannot be fully trusted.

The second technique is an exhaustive test and enables automation. If there is plenty of human resources, then it can be developed in the early stage.

When the software is finished, this technique can be used at once to identify faults.

Both testing techniques relay on the sample data.

If the data is unrepresentative because of any change in the environment,

we may need to collect new data for testing and attempt some validation activity.

R2:

Runtime of the software is obtained by program of timing which is precise and accurate.

The same software may has a change in runtime on different computers because of different cpu performance.

In order to meet the requirement, we need to do extra tests on the computers which will be used for delivery.

2.3

R1：

The first testing approach is a combination of software and inspection. We will need to build scaffolding that automatically produce the combination of flightpath and no-fly zone which is necessary for the inspection.

The second testing approach needs to building scaffolding that testing all the data from the database and do the checking by automation.

R2:

The testing approach can be applied together with the previous scaffolding by adding a timing function.

Both approaches can be applied after the completion of the sofrware so they can start at the same time.

All the scaffolding here deal with duplicate work by automation which saves much time for human.

2.4

The testing plan will work through all the data in the database which gives trustworthly results. It costs lots of time on human part.

It also relay on the reliability of testing data. If the data in the database is not representative, then we need to change the test plan.

Since it is a huge work, using a computer with high level of performance can finish the testing in a short time.

3.1

R1： Drone should not fly over buildings in the no-fly zones

The complexity of orders is similar everyday so no input that are especially valuable.

Random testing is applied. Possible inputs are picked uniformly.

The input for this software is the date so inputs for testing would be 1/1, 2/2 ... 11/11, 12/12.

There is no distinct attribute that can be varied so combinatorial testing is not considered.

technique1 : Produce the flight path and no-fly zone in the same graph and check if it meets the requirement by human.

technique2 : Use the testing software to check if every point in the flight path encounters the no-fly zone.

technique1 : When the flight path and no-fly zone are in the same graph. it is easy to check if the drone encounters the no-fly zone by sight.

The testing software for this technique is easy to desigh. When there is limited resources to do the testing, this technique would be a suitable choice.

technique2 : The advantage of this technique is high accuracy. If the drone meets the no-fly zone for only a little bit, it would be difficult to do the checking by sight.

The testing software can do the checking in terms of coordinate which is more accurate than human.

3.2

technique2 :

The aim of the test is to check if the drone encounters the no-fly zone so we will need to go through each point and check the result.

Coverage is only a proxy for thoroughness or adequacy so both functional test and structure test would be applied.

In order to highlight missed logic, coverage of all branches would be necessary so branch testing is applied.

3.3

In the default map:

technique1 : There is no problem with the flighpath of the drone when checking the synthetic graph with eyes.

technique2 : From the start point to the end poing, no point among the flightpath is identified in the no-fly zone.

branch coverage is 100%.

When the no-fly zones are changed, both techniques identify faults.

In the first technique, points amoung the flight path encounters no-fly zones conspicuously.

3.4

The strategy of generating flightpath is based on the default map so when there is a change in no-fly zones, there should be a large change amoung modules which has a high cost.

The strategy of generating flightpath for varied no-fly zones is not developed due to limited resources.

4.1

Requirement： Drone should not fly over buildings in the no-fly zones

technique1 : Produce the flight path and no-fly zone in the same graph and check if it meets the requirement by human.
It is easy for human to make mistakes and human has bias on almost everything. The synthetic graph may not be precise.
technique2 : Use the testing algorithm to check if every point in the flight path encounters the no-fly zone.
Due to limited resources, no technique is introduced to verify the correctness of the algorithm for testing.

The techniques are based on the sample database.
Data for testing may not be representative, actual orders may bring more barriers for path-finding strategy.
technique2 works on my pc, there is no clue that it can work on the pc for the actual use.
Computation resource is insufficient.

4.2

In all the testing procedures, there should be no case that drone encounters the no-fly zones.
This is an important requirement so there should be be no fault.
Technique2 should have 100% branch test coverage

4.3

In all the testing procedures, no fault was identified for the default no-fly zone.
When the no-fly zone was changed, all the testing procedures failed.

4.4

When the no-fly zones change, adjust the code
This method has high cost in human resources.

Design code for finding pathflight automately
This method can deal with varied no-fly zones which can solve the problem.
Due to limited time, this strategy was not used.

5.1

The first review technique is to check if meaningful variable and function names are used.
All the function names are meaningful. Although most variables have specific names, there are still some variables have single letters as names such as a, b, c.
Because of this review technique, all the variables have a meaningful name finally which makes the code easy to develop and maintain.

The next review technique is to check if the code is properly formatted.
The design of the loops is very complex. Although the software works fine, if there is any error, it takes much time to do the checking.
Due to limited time, the format of the code was not optimized.

The final review technique is to check if all exceptions are explicitly handled.
This technique is very important because unhandled exceptions may cause a system to crash.
Exception handlers were added according to eclipse auto suggestions.

5.2

There are three stages of coding. They are retrieving information of orders from database, designing flightpath and recording the path in another database.
These three stages would be finished in the sequence order.
Early testing is important. The junit tests for each module would be developed at the same time.
When one module is finished, it would be accessed by junit tests at once.
There would be several code integration tests for each stage after all the modules in the stage are done.
In order to avoid integration issues, it is important to frequently merge the code and validate it.
When all the code is finished, functional tests would be applied and performance tests would be executed by using API calls and database queries.

5.3

When junit tests are developed, they would be used at once when the module is finished or changed which could be settled as automation.
Integration tests are complex so these would be activated by human resources.
After the development of software, some functional tests could be carried by automation.
For example, every point on the flightpath should be checked if it encounters the no-fly zones.
Workload is very high for this test so automation would play an important role.

5.4

There are junit tests after each establishment or change of module so any module faults would be identified at once.
For example, I only considered four conditions in the function of getangle, after the junit test, I realized that this function did not cover all the possible situations so I updated it to eight conditions.