Planning and evaluation document

We will select 3 sample requirements from the project requirements document referred to in section 1 of the portfolio for planning and testing. These requirements are:

1. No entering of the no fly zones when calculating a flight path
2. Checking user input dates if they are within the REST server range.
3. Order validation for orders obtained from the REST sever.

Priorities and testing strategies:

**Requirement 1:**

This is an **integration level requirement**, but it is of **high priority**, as it concerns the **safety** of the students or personnel in these crowded areas, the risks and consequences are high if a drone were to fail or malfunction once it is in a busy area. Due to the importance, this requirement needs to be thoroughly tested for a reliable verification of requirements.

The testing approaches may include **partition principle** (we can use functional testing here) and **static testing/redundancy checks** (to ensure bugs and simple errors are caught early), with **manual checks** (such as checking the drone flight path visually on GeoJson) to ensure safety is achieved. This is to achieve the set out at least two of the different **A&T** approaches.

Partitioning principle for functional testing:

To ensure safety, partition requires **exploiting the specification** to selectively test the input space. These often include boundary values where failures are more likely to occur to allow faster fault detection. From the specification, some test boundary values may be that the drone is allowed to touch the no fly zone line but is unable to cross it or that the drone’s movement cannot cross it at any given time despite never directly landing in the zone or one of it’s moves.

Two things to schedule into the plan:

* Some sort of inspection of the path of the drone to see that there are no violation of the specification
* Later more exhaustive tests that checks more combinations of inputs to check it conforms to specification

Static and redundancy checks:

The testing approach complements any tests being carried out on code, it is especially useful for catching faults efficiently due to time constraints.

Manual Checks (System Level):

This approach is after the **functional testing with partitioning of the input space** and also **the static and redundancy checks**. **This system level testing** now requires the actual “real world data” where the REST server may dynamically provide the orders to calculate the flightpath for avoiding the no fly zones. We can plot the generated GeoJson files on the GeoJson website with the all.geojson file provided and manually inspect if the flightpath crosses the no fly zones or not.

From the planned tests, we can see what should be added to the plans to allow for successful test approaches:

* Generate synthetic test case boundary data for no fly zone requirements.
* Scaffolding for the partition functional testing to simulate the no fly zones and a drone flight path option.
* Use real world data from the REST server to allow manual observation checks of the generated GeoJson data file.
* Feeding collected data back into early testing.

**Requirement 2:**

This is a **unit** level requirement. It has a **low priority** compared to the higher level integration level no fly zone requirement. There are no real safety concerns if a fault happens here at the unit level, if an invalid date was entered as the input for retrieval of the REST server and was not caught, an invalid URL exception may be displayed to notify the user of the issue. It is a minor issue.

Testing here may also involve the **partition principle with functional testing**, and also the **static** and **redundancy** checks. Selective input testing can help us program to include those measures.

Partition Principle for Functional Testing:

Selective input values can be checked for validity, for example, the date can be checked using simple unit tests (**assertion checks**) for a simple unit level requirement.

Static Testing and Redundant Checks:

Again, these are checks that are easily done to verify code has no easy faults and can be caught early.

To verify a working system, we need some sort of synthetic data and ways to run tests, the following task may be added to the testing process:

* Generating synthetic data to test the date validation
* No scaffolding is required as this is unit level.

**Requirement 3:**

This is another **Integration** level requirement. It has a **high** priority due to concerns with payments and wrong orders for the customers involved if not correctly implemented. This means that it will have to be rigorously checked for any errors. This time, **partition functional testing, static/redundancy testing** will have to be carried out to achieve at least two of the A&T principles.

Partition functional checks:

**Unit level testing and integration level testing** is now also required, built on top of the Junit unit level testing (checks for retrieving the orders and the validation of order methods) checking of edge case input values (such as same expiry date month as order date checks). Junit will be used to create the simple method tests for edge cases for order validation and integration tests will also be carried out to ensure that the order outcome is labelled correctly with the methods working together.

Decomposing the spec we can see two different tasks that could be added to the plan:

* Ensure that some inspection relating to order with multiple errors labels the order outcome with the correct invalid outcome, and that the requirement is met.
* A later exhaustive test that can check more combinations of inputs relating to the outputs conforms to the specification.

Static Testing and Redundant Checks:

Again, these are checks that are easily done to verify code has no easy faults and can be caught early, and we can also verify the requirement is being met with the **validation and verification** checks using this testing too.

From the planned tests, we can see what should be added to the plans to allow for successful test approaches:

* Generate synthetic test case boundary data for testing of order validity
* Scaffolding for the partition functional testing using synthetic data to simulate the valid and invalid orders test data
* Use real world data collected dynamically from the REST server to test on.

Evaluation and risk of the test plan

Requirement 1:

* No entering of the no fly zones when calculating a flight path

The tasks we have discussed for this requirement may take weeks to complete, as this is such a big task for a small development team. To perform the tasks, we would require a very large amount of data, making sure the restaurant, no fly zone, central area coordinates listed are in fact correct, and that the simulation of the drone movement is accurate enough. This is all simulation data and may not have taken into account factors like wind speed or temperatures affecting the drone performance.

Such a large amount of synthetic data generated may not be useful if the drone movement simulation data is slightly off.

More exhaustive testing can be placed as the Statistical test case generation step of the Cleanroom life cycle, while using GeoJson data to view the drone flightpath violation checks could be considered part of the Formal Design Correctness Verification step of the Cleanroom Life cycle.

Risks here include trying to over simulate and spending too much time collecting data for the drone synthetic data generation. I believe that a redefine of the specifications may be needed or relaxed due to potential software fault causing drone breakdown if the drone was forced into the no fly zone by accidental means. Although seriously hurting someone might prove a big issue as safety is concerned here, the chances are very low and I think the specifications should redefine the no fly zones to be weighted bad to enter but can resort to the option if needed to.

Requirement 2:

* Checking user input dates if they are within the REST server range.

The simple task of adding more synthetic data is quite easy and should only take up to a day to complete. No scaffolding is even required for this task, although if we want even more exhaustive inputs, we could try all the dates available in a year, but it has little to no value with a huge increase in cost if done manually.

The risks are also very low with requirement 2, if we miss some data where the synthetic data was not thorough enough, there are only a small number of days or input possible where faults could happen, where it might lead to some smaller problems like not able to access the website, no one’s safety or privacy is in danger and the chance of it happening in very unlikely if thorough testing has been done.

This task could belong in the Statistical test case generation step of the Cleanroom life cycle, as it is focusing on the generation of the test cases.

Requirement 3:

* Order validation for orders obtained from the REST sever.

This task may take a few days to complete. There are a lot of possible invalid combinations and although the REST sever already supplies some orders, we need to also create our own synthetic data, sometimes, we might not be able to correctly simulate all the ways possible invalid combinations are supplied so real world data may be needed, that could take up to weeks or months if real world data is used instead.

If issues with order validation did indeed happen, this could be a massive issue where either sometime the restaurant does not receive the necessary payment or the user of the service may under or over pay the service. This could cause massive delays and angry customer or restaurant owners. However, chance are quite low and are easily fixed if issues do arise.

This task may be related to the System Test and Acceptance Test step of the SRET life cycle.

Instrumentation and scaffolding

Here we will describe what scaffolding and instrumentation is required to carry out the given tasks.

For our requirements:

Requirement 1:

* Scaffolding of synthetic data generated for the partition test cases and other input cases. Some sort of simulator for the system, to allow the testing of the software
* The use of the system testing later on to allow the visualisation of the drone path to check for violations with geojson data
* The use of the actual data from REST server to allow more thorough drone path calculations testing

Requirement 2:

* No scaffolding is required as this is a simple unit test. Assertion checks are enough here to verify the task.
* The only instrumentation required is the unit test implementation.

Requirement 3:

* Some sort of simulator for the order validation so it is possible to test the software. This is scaffolding and will need to be scheduled early.
* Tests may include REST server data as well as synthetic data for integration level testing.

Evaluation of Instrumentation

The instrumentation provided contains adequate testing as it contains a lot of synthetic data used for unit testing and integration level testing. A lot of real data from the REST server was also combined into testing process. Furthermore, geojson data for different dates of the order period was also generated and manually inspected for any verification faults for requirement 1. The tasks planned and implemented in this document can give us the confidence that most faults will be detected and found.

However, there are still ways to improve the adequacy of testing, such as more synthetic data or asking real users to try and test the services. This would take a lot more time and more unrealistic for a small project with tight budget of time and money.

Perhaps a more realistic simulation for the drone could also help with the synthetic data generated. However, this is unfeasible as that would take a tremendous amount of work that may take a much bigger development team and time.