LO 2

Design and implement comprehensive test plans with instrumented code.

Overall, I would use a DevOps delivery pipeline with continuous development, integration, testing, deployment, and monitoring. Quality assurance should be deeply integrated and the responsibility of all developers. DevOps helps shorten the time between deployments, while offering reliable software at a higher quality.

**Priority and Pre-requisites**

Requirement 1: Customers’ orders must be correctly validated

This is a medium-priority requirement. If invalid orders are considered and potentially added to the final flightpath, there could be a significant waste of time, resources, and revenue. This said, optimistic accuracy is still the overall best choice – it is better to take the risk of wasting some business resources on invalid orders rather than upsetting customers by not accepting and delivering valid orders. Validation is important here; having thorough testing allows the customer to be informed very early in the process that their order is invalid and will not be considered. Generally, the software should be correct and useful to customers – testing should ensure order validation functions are not mistakenly discarding valid orders. Realistic synthetic data should be created with combinations of errors.

Partition, i.e. divide and conquer is a good technique for this requirement. There should be unit testing for each independent validation function, and subsystem structural testing to ensure that the holistic function calls and controls each of these unit functions appropriately. Test cases should cover individual errors and combinations of errors. Combinatorial testing would also be useful, particularly in testing credit card numbers, where multiple prefixes can be valid and combined with a wide range of permutations for the rest of the digits.

Requirement 2: Flightpath module must integrate with no-fly zone module

This is an extremely high-priority safety requirement and must be strong verified against the specification. If the drone makes illegal moves in the final flightpath, it poses not only a physical risk to people in no-fly zones, but breaks Civil Aviation Authority regulations and opens the business up to litigation.

Chapter 3 suggests that a test criterion works best when it is highly sensitive; that is, it is good for faults to cause failure all of the time rather than go undetected sometimes. This is particularly important when calculating whether a drone movement lands inside a no-fly zone. It is easy to test if a movement endpoint is in a polygon, but what if a movement starts and ends outside a polygon, but crosses over the corner mid-flight? It is key that movements like these fail every time. Therefore, it is imperative that sufficient resources are allocated to ensuring this never occurs, regardless of cost to the business.

Unit testing of the flightpath and no-fly zone modules should begin at the earliest opportunity, followed by thorough integration testing. Explicit pass and fail criteria should be set, requiring high levels of reliability, correctness, and safety. Regardless of how algorithm efficiency develops, the flightpath must always be safe; regression tests should be run after each update.

Redundancy is important – parameters and type of data passed between the flightpath and no-fly zone modules must be strictly controlled and checked against in both modules. It is potentially worth recalculating the entire flightpath at the end to ensure the final move-set does not ever fail the criteria. Exhaustive techniques should be used where tests are run multiple times to ensure consistent results.

**Scaffolding and Instrumentation**

Requirement 1: Customers’ orders must be correctly validated

The test plan must include scheduled time for scaffolding the automatic input of synthetic data, both for each unit test plan and the overall structural test plan. Synthetic data can be randomised but needs to cover a wide array of possible inputs categories, including certain valid data, extreme erroneous data, and boundary data.

Structural code coverage analysis of the entire module’s control flow uses static instrumentation, such as calculating the percentage of lines executed when a test is run. This is beneficial as it provides a visual insight into the product quality, and can reveal code paths that are redundant or unnecessary.

Requirement 2: Flightpath module must integrate with no-fly zone module

Scaffolding for this requirement needs to be extensive. First, for unit testing the flightpath module, a stub of the no-fly zone module is required to provide analysis of which drone moves are illegal. Without the actual module, this might just involve random selection and testing that “illegal” moves are correctly not added to the flightpath.

Regarding the no-fly zone calculations, there are a large variety of no-fly zone polygons that may be added in the future, and they could be any manner of shapes. The algorithm must be tested sufficiently to ensure it copes with convex and concave polygons, and polygons with holes. A wide array of randomised polygon data is needed alongside a wide array of randomised drone movement positions.

Regarding the flightpath algorithm, scaffolding involves building realistic order data sets which can bring the drone extremely close to no-fly zones.

Furthermore, instrumentation in the form of non-functional performance testing can be used here to satisfy the requirement that the flightpath calculation must not take more than 60 seconds. Timing the overall processing time for one move, how long the algorithm takes to choose a next move, and how long the no-fly zone module takes to determine whether a move lands inside a polygon, can provide insight as to the additional time spent integrating these functions together.

**Process and Risk**

Requirement 1: Customers’ orders must be correctly validated

The validation functions in question are fairly simple and independent of each other. With multiple developers, it would be possible to develop and unit test these with smaller individual sets of synthetic data concurrently.

Integration / control-flow testing of the whole validation class requires these unit functions to be completed and tested.

Creating synthetic order data that successfully covers the range of valid, erroneous, and boundary categories will be lengthy but can be automated to some degree.

These functions can be tested at a lower priority in early stages of the process, as while they are important for validation, they are not safety-critical.

Regarding risk, these functions are still very important to the final product and so any delay in generating tests and synthetic data will still have a significant impact on the final deployment date.

Requirement 2: Flightpath module must integrate with no-fly zone module

Strict tests for this requirement should be a critical part of the lifecycle from the early stages. Building tests that ensure robustness and safety is likely to take a long time.

As the efficiency of the flightpath algorithm develops, regression tests should be run to ensure it remains safe. This will take a lot of time and computational resources, but if automated, can be run in parallel with other tasks.

Accurate testing of the flightpath module will require the no-fly zone module to be finished.

Full integration testing of these modules will require a vast amount of synthetic data, including plenty of illegal moves. Generating representative data is likely to be a complex and lengthy task. It is not guaranteed that randomly generated data will provide realistic problems such as where a movement cuts the corner of a no-fly zone, so validation testing is required to ensure representativeness of the training data. If data is not representative, safety cannot be guaranteed, and as safety is a critical aspect then final system integration testing and deployment will be heavily delayed. Delayed deployment means paying for human and computational resources for longer on top of a lack of revenue.

Generally, a risk of DevOps is that widespread automation requires good configuration. If this is not present, errors can occur and unrepresentative tests can propagate incorrect assumptions throughout the process.