Testing Plan

1. **Chosen set of requirements**
    a) **List of requirements to be tested**
        i. R1: The system must ensure that drones do not fall from the sky during the delivery.
        ii. R2: The system must ensure drones never enter any no-fly zones at any time.
        iii. R3: The system must validate that each drone has a valid service-point mapping: a drone ID must appear in at least one servicePointDrones entry, and the corresponding servicePointId must exist in the servicePoints list.
        iv. R4: The system must return the calculated delivery path in three minutes.
            1. R4' (restricted verification proxy): 1 minute on local environment (cost-effective proxy)
    b) **Justification**
        i. All the requirements except R3, which is more at the integration level, are system-level requirements. The first three requirements focus on ensuring the system behaves as expected. R1 is a critical robustness functional requirement that ensures the system is safe. R2 is a functional requirement which regulates the path planning, ensuring the system is correct. R3 is also a functional requirement, which ensures the system is correct by never allowing a drone with an invalid service-point mapping to be used. R4 is a measurable quality attribute that focuses on boosting system performance. Together, these requirements let the test plan explore functional safety, correctness, integration behaviour, and quantitative performance/accuracy.
2. **Priority and Pre-requisites**
    a) **Assessment of the requirements' A&T needs:**
        i. **R1**: This is a safety requirement that ensures the system won't cause any potential damage by letting drones crash into people or properties.
            1. We should allocate sufficient resources to ensure the final system meets this requirement.
            2. The completeness property of the quality process suggests that we use appropriate activities to detect each important class of faults, and in this case, it is functional category-partition testing.
            3. The timeliness property of the quality process suggests that we start testing critical requirements as early as possible to reduce T&A effort.
            4. The sensitivity principle states that the tests we select should all

agree on whether the system fails. In this case, we should only include test cases that verify that the total move along the calculated drone delivery path is less than or equal to the drone's maximum move.

5. The redundancy principle suggests that we use different test methods and use assertions together with testing. Since the primary test method selected for this requirement was functional category-partition testing, a complementary testing method could be statistical analysis, but that would cost too much effort and resources. The cost-effectiveness property of the quality process suggests that we choose activities based on cost, which means the best way to meet this principle is to use assertions alongside testing.
6. The partition principle suggests that we handle the testing and verification by suitably partitioning the input space. For R1, the input space depends on the lists of drone bases, drones and their availabilities, and dispatch orders. This results in a complex input space. But, as mentioned in the LO1 requirements document, we can build our partitions based on the relative positions and numbers of orders, so that the system can return a valid path if the number of orders is reasonable and they are not too far apart. We then have the following factors
    a) Total distance of the calculated delivery path
        i. D1: Total path distance is well below the maxMoves field of the drone being used
        ii. D2: Total path distance is approximately equal to the maxMoves field of the drone being used
        iii. D3: total path distance is greater than the maxMoves field of the drone/drones being used, but is splitable across multiple trips/drones
        iv. D4: Total path distance is a little bit greater than the maxMoves field of any drone and not splitable.
        v. D5: Total path distance is much greater than the maxMoves field of any drone and is not splitable.
    b) Number of orders
        i. N1: Single order
        ii. N2: multiple orders
7. Branch coverage will be used to assess the quality of the test suite. The target coverage is above 80% for the method calcDeliveryPath(),

as this is the method where the path planning and move calculations are done.

8. Based on the former sections, we need to schedule the following activities
    a) Adding assertions to the code to ensure that once the total moves of a drone exceed its maxMoves field in the calculated delivery path, the system immediately stops.
    b) A later exhaustive test, in which each test case corresponds to a partition derived from the factors described above. This will require writing some scaffolding code for the test cases.

ii. **R2**: This is an essential functional requirement which ensures the system is correct and makes sure the drones are operated in the allowed areas.

1. Our customer probably wants this requirement met strictly because there may be regulatory policies for the drone, which suggests we dedicate sufficient resources (at least at the same level as R1).
2. The completeness property of the quality process suggests that we use appropriate activities to detect each important class of faults, and in this case, it is functional category-partition testing.
3. The timeliness property of the quality process suggests that we start testing critical requirements as early as possible to reduce T&A effort.
4. The sensitivity principle states that the tests we select should all agree on whether the system fails. In this case, we should only include test cases that verify that any coordinates in the JSON object returned by the system are not in any no-fly zones.
5. The redundancy principle suggests that we use different test methods and use assertions together with testing. Since the primary test method selected for this requirement was functional category-partition testing, a complementary testing method could be statistical analysis, but that would cost too much effort and resources. Using assertions is also not feasible, because for each coordinate, we would have to loop through all no-fly zones to check whether it is within any of them. The cost-effectiveness property of the quality process suggests that we choose activities based on cost, which means we should only do testing for this requirement.
6. The partition principle suggests that we handle the testing and verification by suitably partitioning the input space. For R2, the input space depends on the lists of no-fly zones, drone bases, and dispatch orders. This results in a complex input space. But, as

mentioned in the LO2 requirements document, we can build our partitions based on the number and positions of no-fly zones, and the relative positions of the delivery points and drone bases. So, we can consider the following factors.

a) Position of no-fly zones relative to the "straight-line" route between service points and delivery points
   i. RA1: No no-fly zones.
   ii. RA2: No-fly zones exist, but do not intersect any shortest path
   iii. RA3: A Singular no-fly zone intersects the shortest path, but alternative safe paths exist.
   iv. RA4: Multiple no-fly zones intersect many potential paths, which makes the alternative paths complex
   v. RA5: No-fly zones block all paths between the service points and at least one delivery point

7. Branch coverage will be used to assess the quality of the test suite. The target coverage is above 80% for the method calcDeliveryPath() (same as R1), and above 95% for the method aStarSearch(), as this is the method where the system creates actual fly paths.
8. Based on the former sections, we need to schedule the following activities:
   a) A later exhaustive test, in which each test case corresponds to a partition derived from the factors described above. This will require writing some scaffolding code for the test cases.

iii. **R3**: This is another important functional requirement which improves the robustness of the system.

1. Since this requirement is less important than the former two (not meeting this requirement won't raise safety concerns nor violate legal regulations), we can put less effort into it.
2. The completeness property of the quality process suggests that we use appropriate activities to detect each important class of faults, and in this case, it is functional category-partition testing.
3. The timeliness property of the quality process suggests that we start testing critical requirements as early as possible to reduce T&A effort. But since this requirement is less important, we should do this after finishing testing R1 and R2.
4. The sensitivity principle states that the tests we select should all agree on whether the system fails. In this case, we should only

include test cases that verify that if a drone has a valid availability mapping.

5. The redundancy principle suggests that we use different test methods and use assertions together with testing. Since the primary test method selected for this requirement was functional category-partition testing, a complementary testing method could be statistical analysis, but that would cost too much effort and resources. The cost-effectiveness property of the quality process suggests that we choose activities based on cost, which means the best way to meet this principle is to use assertions alongside testing.
6. The partition principle suggests that we handle the testing and verification by suitably partitioning the input space. For R3, the input space depends on the lists of drone bases, drones and their availabilities. As mentioned in the LO1 requirements document, we can build our partitions based on whether the drone has a valid service point mapping. We then have the following factors
    a) Mapping status of a drone:
        i. M1: The mapping is valid. The drone ID appears in at least one element of the list servicePointDrones, and the corresponding servicePointId exists in the servicePoints list.
        ii. M2: The mapping is not valid. The drone ID appears in at least one element of the list servicePointDrones, but the corresponding servicePointId does not exist in the servicePoints list.
        iii. M3: The mapping is not valid. The drone ID does not appear in any element of the list servicePointDrones.
    b) Input composition
        i. C1: All drones are M1
        ii. C2: There is at least one drone that is M2.
        iii. C3: There is at least one drone that is M3.
        iv. C4: There is at least one invalid-mapped drone (M2 or M3) and at least one valid-mapped drone (M1)
7. Branch coverage will be used to assess the quality of the test suite. The target coverage is above 80% for the method calcDeliveryPath() (same as R1), and above 95% for the method getServicePointPosition() as this is the method for finding a drone's servicepoint.
8. Based on the former sections, we need to schedule the following

activities:

a) Adding assertions in the planning loop to check if a drone has a valid service point on a given date and a given time
b) A later exhaustive test, in which each test case corresponds to a partition derived from the factors described above. This will require writing some scaffolding code for the test cases.

iv. **R4**: This is a measurable quantitative attribute which focuses on improving the user experience

1. This requirement is the least important of the four. Failing this requirement won't cause any safety or correctness concerns. Not meeting this requirement will only slightly affect the user experience. It is suggested that we put the least effort into testing if the system meets R4.
2. We can only test this for the completed system, so we should leave this system test to the end of the whole A&T process, even though the timeliness property of the quality process suggests that we start testing critical requirements as early as possible to reduce T&A effort.
3. We can easily test this requirement by measuring the latency between sending the request and receiving the response.
4. We need a tool like IntelliJ's built-in latency measurement tool to accurately measure latency.
5. In a real-world deployment, latency may increase significantly with network traffic. It would take a lot of time and money to simulate a real environment for this system.
6. The cost-effectiveness of the quality process suggests we choose activities based on cost. Also, the restriction principle suggests that suitable restrictions can reduce hard problems to simpler problems. As a result, we can further constrain R4 for a local running environment. Thus, our new requirement is actually R4.
7. Based on the former sections, we need to schedule the following activities:
    a) For each test case we built for R1, R2 and R3 (14 tests in total), we ran 5 times and recorded the latency. And for each test case, we report:
        i. The max latency
        ii. The median of the latency
    b) Then, to connect our data together, we calculate:
        i. The overall max, which is the maximum latency of all the

maximum latencies of each test case.

ii. The interquartile range of the median latencies for the test cases.

b) **Scaffolding and Instrumentation**

i. **Scaffolding plan**

1. **R1**: Since we need to add assertions to the code and write exhaustive test cases, this will require some scaffolding. Since the system needs to request data from another remote server, and we cannot modify the data stored on that server, we would need scaffolding to mock the remote server and serve the test data we need.
2. **R2**: For this requirement, we need to write some scaffolding code for two things: the test cases themselves and the mocking of the remote server that serves the data
3. **R3**: Similar to R1, we need to write scaffolding code for three parts: the assertions, the test cases themselves and the mocking of the remote server.
4. **R4**: We don't need any scaffolding code here because we are simply re-running our tests and recording the latency using the timer built into our IDE.

ii. **Evaluation**

1. The scaffolding for R1 should be adequate. The assertions should supplement the test cases and stop the system once maxMoves is exceeded, ensuring the system is safe. The planned test cases and mock objects are also necessary to meet the requirement. To further improve, error logging can be added to make debugging easier, as all error status codes (such as 500) are mapped to 500 and error messages will not be displayed on the terminal.
2. The scaffolding for R2 focuses on mocking and building test cases because checking every coordinate with assertions is expensive, as described. However, the checking can be added back to the system if there is plenty of time left by the end of the A&T process.
3. Similar to R1, the scaffolding should be adequate. The assertions can be easily implemented; mocking and test cases are also necessary.
4. For R4, no scaffolding is needed, so this one should be fine.

c) **Process and Risk**

i. For testing purposes, we will use the DevOps lifecycle, in which testing occurs after development. DevOps fits into this test plan for the following reasons. First, we make a plan before development, which includes

development and testing, and this is precisely what we are doing right now. Second, testing happens after the software is created, which is the stage where we are now. We have constructed our testing plan and have the software to be tested ready; we need to conduct the actual A&T process.

ii. There are a few risks we need to consider. (Evaluation of test plan quality)

1. First, since this test plan is heavily test-based, our test results would be optimistically inaccurate, as indicated by the pyramid diagram in Chapter 2. We can't fully mitigate this risk because we aren't using statistical analysis due to cost-effectiveness. But by carefully partitioning our input space, we should be able to have partitions that somewhat cover all cases.
2. There should be no technology-related risk in this plan, as it arises from unfamiliarity with the tools being used. In this plan, only two tools will be used: the IDES (IntelliJ), which I am already familiar with during development. So there are no technology risks.
3. There might be some schedule risks. Since the test order is R1, R2 -> R3 -> R4, and R1 and R2 together generate many test cases, there is a risk that R3 and R4 will not be tested thoroughly. However, as described above, R4 is not a very important requirement, so even if it is not tested thoroughly, the quality won't be affected too much. On the other hand, we want to ensure that R3 is met because it improves the system's robustness. To mitigate this, we can carefully select a set of presentational test cases for R1 and R2 (Note that when doing this, we need to make sure the first risk is also mitigated, i.e. we shouldn't reduce the number of test cases drastically.) so that we leave enough time to test for R3.
4. The schedule risk can also arise from inadequate unit testing. In this testing plan, R1, R2, and R4 are system-level requirements, and only R3 is an integration-level requirement. This means there is no unit-level requirement, and the final test result may be affected by inadequate unit testing. The best way to mitigate this is to add unit tests. And since this is a large system, the endpoint for calculating the delivery path uses only part of the system's functions, reducing test effort. Also, since all those requirements point to a large function which is responsible for calculating the delivery path, not having unit tests should not cause a problem.