LO3: Apply a wide variety of testing techniques and compute test coverage and yield according to a variety of criteria

3.a. Range of techniques

The implemented tests use multiple techniques across unit, integration and system levels, consistent with the approaches identified in LO1.

Unit testing

Core domain functions (geometry and validation) were tested using specification-derived partitions:
Equivalence Class Partitioning (ECP): typical valid inputs vs invalid inputs (e.g., null coordinates, invalid angle, malformed polygons).
Boundary Value Analysis (BVA): coordinate limits (±90, ±180), angle boundaries, and minimum polygon sizes.
These unit tests cover both correct path behaviour and robustness behaviour (MR1), and they provide strong identifiers because expected results are expressly defined as a certain number.

Integration testing

Integration tests check that the external ILP service retrieval and mapping logic works correctly (FR6). This includes mocking the upstream boundary (e.g., ILP REST interactions) and verifying that DTOs are correctly converted into the project's internal entities.
This is interface-focused integration, aimed at catching schema/assumption errors early.

System testing

End-to-end API tests (MockMvc) check the planner endpoints through complete HTTP request/response flows:
calcDeliveryPath returns totals and a plan structure (FR9).
calcDeliveryPathAsGeoJson returns valid GeoJSON (FR10).
Determinism checks (MR4) ensure repeated identical requests return identical results in key fields.
A timing test provides limited non-functional evidence for MR3 (treated as environment-dependent).

Structural/coverage-driven testing

IntelliJ coverage reporting was used as a structural evaluation technique.
When coverage highlighted complex logic in DroneService as under-exercised, targeted unit tests were added to cover missing branches (e.g., operator branches, availability filtering branches, droneDetails error path, queryAsPath, and multi-delivery planning paths). This shows coverage was used as feedback to evolve the test set.
Overall, the project demonstrates a broad mix of tests, including functional tests, integration tests, system tests, and structural coverage measurement.

3.2 Evaluation criteria for adequacy of the testing

Testing adequacy is evaluated using multiple criteria:

(1) Requirement-based adequacy is first judged by whether the chosen requirements from LO1 have direct test evidence:
   - Unit-level correctness/robustness (FR2, FR4, FR5, MR1, MR2)
   - Integration at the ILP boundary (FR6)
   - System-level end-to-end planner behaviour and output format (FR9, FR10, MR4)
   - Timing regression indicator (MR3)

   This gives direct evidence that the test set is aligned with the required behaviour.

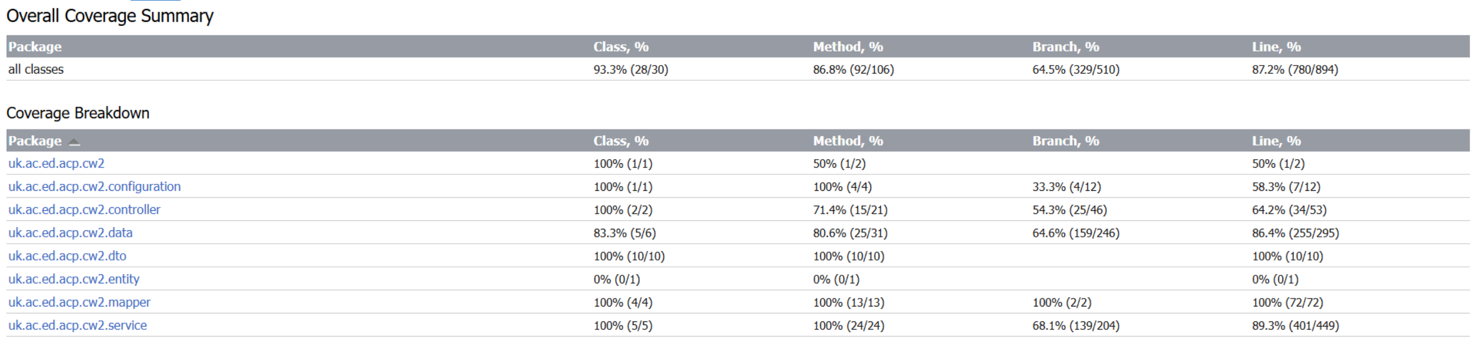
(2) Structural adequacy (IntelliJ line and branch coverage) IntelliJ coverage provides a quantitative check of how much code and decision logic executes during the test suite. This matters because large parts of this project are branch-heavy (filters, validation, query operators, planner control paths).

(3) Thirdly, adequacy was assessed across multiple testing levels. Unit tests were used to validate local behaviour of core domain logic (e.g., geometry calculations, validation rules, and query operator handling) in isolation. Integration tests were used to confirm correct interaction between components and external boundaries (in particular, the ILP retrieval/mapping boundary), ensuring that assumptions about request/response structure and mapping held when subsystems were combined. System-level tests were used to verify complete workflows through the HTTP API, including planner endpoint behaviour. Using multiple levels reduces the risk of faults being missed due to relying on a single testing approach.

(4) Lastly, test execution results were used as a supporting indicator. Successful execution of the automated test suites suggests that the expected code paths were exercised and behaved as intended for the scenarios covered. These results are treated as indicators rather than proof of correctness, since tests only demonstrate correctness for the specific inputs and execution paths included in the test set, and untested edge cases may still exist.

3.3 Results of testing

All tests run successfully, and IntelliJ produces measurable coverage evidence. A link to a pdf version of the coverage summary can be found here in the portfolio repository. A full report can be found named index.html here.

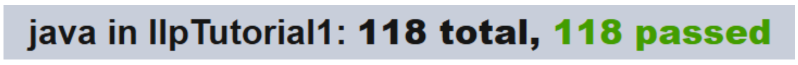

Overall Coverage Summary

Package	Class, %	Method, %	Branch, %	Line, %
all classes	93.3% (28/30)	86.8% (92/106)	64.5% (329/510)	87.2% (780/894)

Coverage Breakdown

Package	Class, %	Method, %	Branch, %	Line, %
uk.ac.ed.acp.cw2	100% (1/1)	50% (1/2)		50% (1/2)
uk.ac.ed.acp.cw2.configuration	100% (1/1)	100% (4/4)	33.3% (4/12)	58.3% (7/12)
uk.ac.ed.acp.cw2.controller	100% (2/2)	71.4% (15/21)	54.3% (25/46)	64.2% (34/53)
uk.ac.ed.acp.cw2.data	83.3% (5/6)	80.6% (25/31)	64.6% (159/246)	86.4% (255/295)
uk.ac.ed.acp.cw2.dto	100% (10/10)	100% (10/10)		100% (10/10)
uk.ac.ed.acp.cw2.entity	0% (0/1)	0% (0/1)		0% (0/1)
uk.ac.ed.acp.cw2.mapper	100% (4/4)	100% (13/13)	100% (2/2)	100% (72/72)
uk.ac.ed.acp.cw2.service	100% (5/5)	100% (24/24)	68.1% (139/204)	89.3% (401/449)

A link to the pdf of all tests and their outcomes can be found in the portfolio repository here.


java in IlpTutorial1: 118 total, 118 passed

This is strong structural evidence that high-risk, complex business logic has been exercised in the test suite.

3.4 Evaluation of results

Strengths/confidences

Very strong confidence in domain-level correctness due to:

- high line coverage overall (87.2%) and strong coverage in service and data, where most complex business logic is held
- systematic functional input testing (ECP/BVA) with clear correctness identifiers for geometry and validation

Strong confidence in planner endpoint behaviour and interoperability:

- system tests validate required output structure and GeoJSON structure validity, not just internal methods

- determinism checks increase confidence in repeatability (MR4)

Strong evidence that coverage was used as an evaluation tool:

- DroneService is an inherently complex, branch-heavy class; achieving 90.6% line and 66% branch means the test suite exercises most major decision logic in that class and reduces the likelihood of untested logic faults.

Limitations/residual risk

- Branch coverage is still lower than line coverage. This indicates that some alternative paths (rare error conditions, unusual combinations of dates/times, extreme planner cases) are still not executed.
- Coverage is not proof of correctness, as executing a branch does not guarantee the branch's behaviour is correct, only that it ran.
- Lower coverage in controller and configuration packages, which indicates that not all controller-level and configuration-level behaviour is validated end-to-end; this is a common trade-off when focusing tests on domain logic and deterministic system behaviour.
- Entity package shows 0% coverage for at least one class. This is due to it being a simple data holder, not directly invoked by code paths under test. If required, additional tests could explicitly construct/use that entity, but the risk reduction value may be small compared to testing behaviour-heavy services.

Overall, the results show a strong and diverse test suite with measurable structural evidence. The remaining gaps are identifiable (branch-heavy edge paths; controller/config paths), and these limitations can be justified as a scope/resource trade-off.