Learning Outcome 5 – Conduct Reviews, Inspections, and Design and Implement Automated Testing Processes

5.1 Identify and Apply Review Criteria to Selected Parts of the Code and Identify Issues in the Code

The project code was inspected using both automated static analysis (via IntelliJ IDEA) and manual peer review sessions. Review criteria included naming conventions, method length, cohesion, and adherence to the single responsibility principle. Notable issues were identified in large conditional structures in RegionService and inconsistent use of early returns in DroneAvailabilityService.

In ValidationService, certain repeated validation blocks were refactored into helper methods to reduce duplication. Code structure was evaluated for clarity, with service responsibilities clearly separated across modules like PathfindingService, DeliveryPathService, and DroneAvailabilityService.

Although Checkstyle or SpotBugs were not configured, IntelliJ inspections helped enforce a consistent style, highlight unused imports, and identify unreachable code. Future enhancements could include integrating a dedicated static analysis tool with stricter rule enforcement.

5.2 Construct an Appropriate CI Pipeline for the Software

A basic CI process was implemented using Maven's default lifecycle phases, triggered via local pre-push checks. The process included build validation, compilation, unit test execution via Surefire, and JaCoCo instrumentation for coverage reporting.

The workflow ensured that all tests passed and updated coverage reports were generated after every significant code change. While the project did not utilize GitHub Actions or containerized CI environments, the structure adheres to CI principles and can be extended to remote execution with minimal configuration.

The CI structure followed this progression:

1. **Code Commit** – Version control via Git tracked all changes.
2. **Build** – mvn clean verify compiled code and ran tests.
3. **Test** – Surefire ran all 143 unit and integration tests.
4. **Coverage Report** – JaCoCo produced updated instruction and branch coverage summaries.

This pipeline provided confidence that regressions were caught early and feedback on coverage gaps remained visible throughout development.

5.3 Automate Some Aspects of the Testing

Test execution was fully automated via Maven, with test cases defined for each service module and controller. Unit and integration tests ran on every build using the Surefire plugin, enabling quick validation across modules.

Instrumentation with JaCoCo automatically collected metrics on instruction and branch coverage. Reports were generated in HTML format and reviewed locally after each test phase. Exploratory testing and robustness checks were added for endpoints such as /calcDeliveryPath and /isInRegion, and assertions validated service logic under normal and abnormal conditions.

Although random input testing and fault injection were not implemented, the framework allows for future automation enhancements. These could include dynamic payload variation, load testing with JMeter, or mutation testing for resilience validation.

5.4 Demonstrate the CI Pipeline Functions as Expected

The Maven-based CI pipeline successfully validated the software's build integrity and testing coverage at every development milestone. All 143 tests consistently passed across modules, and no test regressions were observed during iterative development.

Coverage results (~60% instruction, ~38% branch) were updated automatically after test runs. Local HTML reports enabled review of test gaps without requiring external hosting or workflows. While no full deployment pipeline or GitHub Actions workflow was implemented, the existing Maven process functioned as a local CI approximation and maintained code quality during development.

Going forward, the pipeline can be extended to include containerized environments, automated deployment, and continuous monitoring. For the current project scope, however, the implemented CI was sufficient to detect issues early, enforce testing rigor, and maintain high confidence in code correctness.

This document outlines the automated testing processes and CI techniques applied in the drone delivery project. It highlights code review outcomes, describes the testing pipeline, and evaluates automation and verification workflows in place.