Software Testing 2025/2026 – Continuous Integration (LO5)

Construction

The CI pipeline would carry out tests and publish their results on each commit to the project's repository. Here is a breakdown of the steps included in this pipeline:

1. Static checks – run basic checks that do not require code to be executed, such as identifying deviations from coding style (enforced by a package such as Checkstyle) as well as syntax errors and dead code (detectable by SpotBugs)

2. Unit tests – execute all **fast** unit tests e.g. those in R1.1 and R1.2, using stubs for synthetic map, restaurant and zone data. These should meet the structural and constraint/map coverage targets detailed in the Test Performance document.

3. Integration test – run integration tests using mock endpoints (via WireMock) and tests that require interaction between components e.g. R1.2 tests that require order validation. Additionally, it should be tested that data is retrieved properly from the real HTTP endpoints.

4. Performance tests – carry out a subset of R2.1 faster performance tests using synthetic benchmark maps. A time budget should be enforced to ensure fast feedback when committing – hence the faster tests.

5. Reporting – test results, coverage metrics and performance should be reported. Quality gates will be implemented to fail the build should they not be met by these criteria

Automation

Most of the project's test suite can be automated as part of the CI pipeline.

- Order validation tests can be fully automated using JUnit – can run on every commit
- Path validity tests can be automated using map generators and the existing route oracle – allows batch testing to be run unattended
- Performance tests can be partially automated – the less resource-intensive tests can be run on every commit, whereas more extreme cases can be run regularly at longer intervals e.g. daily

As mentioned above, a time budget should be allocated to these tests to ensure fast feedback. They also must be deterministic for consistency across changes. Results should be archived to allow for regression error analysis

Pipeline Functionality

The pipeline provides early automated testing for behavioural, functional and performance regression. Here are some examples of issues it would identify in its implementation:

Pipeline component	Issues identified	Examples
Static checks	Code quality issues	Unused variables, deviations from code style will be flagged
Unit tests	Logic errors in validation, path generation	If a change to validation causes invalid errors to be accepted, R1.1 tests will fail.

		If a change to path generation allows the path to start in an NFZ, R2.1 tests will fail
Integration tests	Change in endpoint contract	If zone or restaurant JSON format changes, integration tests will fail
Performance tests	Performance regression	If a new pathing algorithm increases runtime, the build will fail