LO 5

Conduct reviews and 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.**

Not completed.

**5.2 Construct an appropriate CI pipeline for the software.**

A continuous integration pipeline aims to provide various automated services to developers, including quick, standardised feedback loops on how well the software passes tests and criteria.[1](#_https://www.tutorialworks.com/cicd-)

The CI pipeline would contain the following stages, all of which are automated:

1. **Automatic trigger, either from code commit or scheduled workflow**

Whenever a developer pushes new code to the source code repository, the pipeline automatically begins the following process.

The repository will be stored on GitHub, which is a version control system allowing for multiple developers to work on a project at once via their own branches.

1. **Build**

Compile the project using the build tool Maven in a clean environment.

1. **Unit tests**

Execute the stored tests via Maven. (Tests are written with JUnit, a test execution framework.)

Verify that all unit tests still pass.

1. **Packaging**

Package the project into a JAR file.

1. **Acceptance tests**

Execute stored JUnit acceptance tests.

Verify that all acceptance tests pass.

1. **Deploy**

Deploy to a staging environment for the production team to check over before deploying to production.

**5.3 Automate some aspects of the testing.**

The main thing requiring automation is the generation of synthetic data. In terms of the chosen requirements, this includes a wide array of orders, both invalid and valid; the same orders can potentially be fed into the flightpath algorithm as well. This could be done by Java’s Faker API, which creates pseudorandom data according to regex.[2](#_https://www.baeldung.com/java-faker) Automatic data generation allows for a far vaster test suite size than would be available by hand.

Secondly, the actual execution of these tests can be automated. The project is in Java and the test framework JUnit provides automated testing. Acceptance testing can also be automated with JUnit, although it would require creating an internal Domain-Specific Language (DSL).[3](#_https://www.infoq.com/articles/inte) This is a way of making automated acceptance tests quicker to write and far more human readable.[4](#_https://medium.com/@bas.dijkstra54/) However, creating a DSL would require a lot of further resource investment.

Furthermore, regression testing will be automated by the pipeline. That is, it will ensure standards are not lost – only improved – as the product becomes more and more developed.

**5.4 Demonstrate the CI pipeline functions as expected.**

A major focus of the CI pipeline is regression testing.

As specifically mentioned in previous LOs, this relates critically to the calculations of the flightpath and how it must not include any illegal moves. It is likely that new flightpath algorithms will be created over time with higher efficiency and other benefits. Regression testing ensures safety standards are maintained whenever a new flightpath algorithm is committed to the repository.

Having such large test suites automated allows for quick analysis of the current state and quality of the product. The pipeline might show a lot of unit test errors centered on a specific module, or generated from specific use cases that human testers might not have thought of. In this way, it provides insight as to quality gaps and where immediate improvements are required. Particularly, auto-generated data can reveal rare boundary use cases, and adapting to these early on prevents wasting resources later down the line. Generally, continuous integration makes it easy to grow products gradually and fix-as-you-go.

A way of telling the pipeline is functioning as expected is that, over the course of the project, the categories of errors naturally change. At first, unit tests should be the primary category requiring patches, but as these gradually get resolved across commits, integration and finally system tests should be the main categories of errors. This demonstrates the progress and quality growth.

**References**

## <https://www.tutorialworks.com/cicd-pipeline-stages/>

## <https://www.baeldung.com/java-faker>

## <https://www.infoq.com/articles/internal-dsl-java/>

## <https://medium.com/@bas.dijkstra54/making-test-frameworks-readable-the-domain-specific-language-c154c9a9abcb>