LO5 - Software Testing

B196275

January 23, 2024

1 Code Inspection

Inspection will contain the following stages: Preparation, Review, and Follow-Up. I am the only inspector for any code reviews carried out as employing classmates would be considered as academic misconduct since the software was built for coursework.

No reward mechanisms or incentives were required, I was mainly motivated by achieving the highest mark possible, thus I took a meticulous approach to code inspections.

Preparation: The artifacts to be inspected were ready before the code review. As mentioned in the previous paragraph, I am the sole inspector, and thus I am responsible for carrying out activities relating to the code inspection. Inspections are normally carried out for each iteration and at the end of development.

Review: I employed a systematic approach to inspection, with 2 checklists that cover different levels of abstraction, one that is high-level, and one that is low-level. Further, I created a use-case checklist for the order validation. Please refer to `inspection.pdf` for the table of results for the code reviews.

Follow-Up: Some attributes relating to maintainability were not met, however, due to this not being an important requirement for our software to possess this. There is further documentation on `inspection.pdf` for each attribute that didn't pass the code inspection.

- **Summary**: Certain components failed the code inspection however these were small issues relating to maintainability. There was no need for examples or usage instructions in the code either or for comments on imports.

From the summary, we can determine that we don't need to carry out the code inspection again, there was nothing major that would affect the functionality of our code, which is what I am prioritising.

2 CI/CD Pipeline Design

2.1 Development

This stage can potentially fail if there are issues with pushing the code to GitHub for trivial reasons.

- Development was carried out by myself as the sole developer. The language used was `Java 18`, and the IDE used was `IntelliJ`.
- Code was committed and pushed to main on `GitHub` via the command line. This will trigger the CI Pipeline. If I were to hypothetically use a CI pipeline to develop this software, I would push it after completing the development of every new feature.

2.2 Build

This stage can potentially fail if there are syntax errors or compile time errors (index out of bound, null pointer exception, etc), it may not build.

- At this stage `Github Actions` is used to clean and build the artifact before running the tests.

2.3 Test

This stage could fail if any of our unit or integration-level tests fail, meaning that the code is not functioning as expected.

- Tests are carried out by the "mvn test" command, the previous build stage is triggered by this command and then the tests are automatically run.
- The tests used were written with `JUnit`.

2.4 Deploy

This stage could fail if `Docker` servers are down, however, this is out of our control, and most likely won't be an issue.

- The repository is then deployed to a `Docker` image for consistency and is pushed to the main branch.

2.5 General Comments

I used this design to build a CI/CD pipeline to help with the CI/CD pipeline demo in Section 4 of this document. It makes it simpler for me to outline what happens in different cases.

3 Test Automation

Unit, Integration and System level testing should be automated when a developer pushes the code to `GitHub`.

Unit Tests: A high number of unit tests should be automatically carried out to verify each component functions as expected, and detect any algorithmic faults that could have potentially arisen from changes made during development.

Integration Tests: Integration tests should be carried out automatically to verify that different components still correctly integrate after any changes are made. In this case, it would be useful to check that our application is still integrating with the REST server properly, especially as it is so crucial to the functionality of the application.

System Level Tests: Some system-level tests should be automated to verify that the application outputs the correct files given certain inputs. At this stage, there shouldn't be any algorithmic faults being detected, as these should have been caught in the unit tests.

General Comments: Tests will be automated in the following order, unit, integration, and then system.

4 CI/CD Pipeline Demo

4.1 Scenario 1 - Successful

This is a walkthrough of a successful push to the CI/CD pipeline, with an explanation of why the push was expected.

Process Walk through:

- Code is pushed to `GitHub`, and then the project is built. In this case, there are no compile-time errors thus it successfully builds.
- The tests all automatically run on the built project, and in this case they would all pass.
- After this the project is packaged into an *über* JAR using `mvn pkg`.
- The repository is then deployed to a `Docker` image.

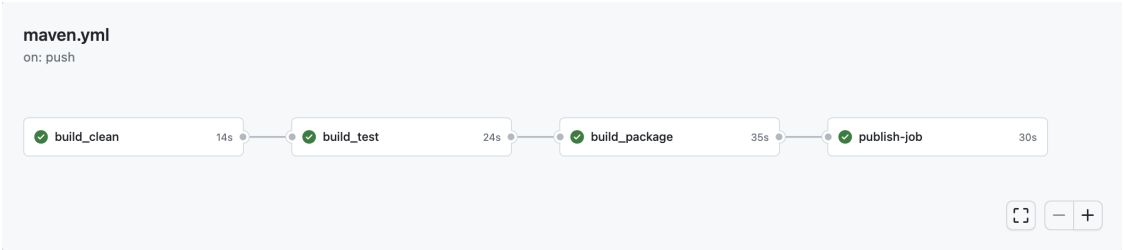

Figure 1: Every stage of the CI/CD pipeline passing

4.2 Scenario 2 - Test Case Fails

This is a walkthrough of an unsuccessful push to the CI/CD pipeline, with an explanation of why the push failed, and the repository wasn't deployed to a docker image.

Process Walk through:

- Code is pushed to `GitHub`, and then the project is built. In this case, there are no compile-time errors thus it successfully builds.
- The tests all automatically run on the built project, and in this case they don't all pass, thus the push is rejected and we do not move onto the next stage.

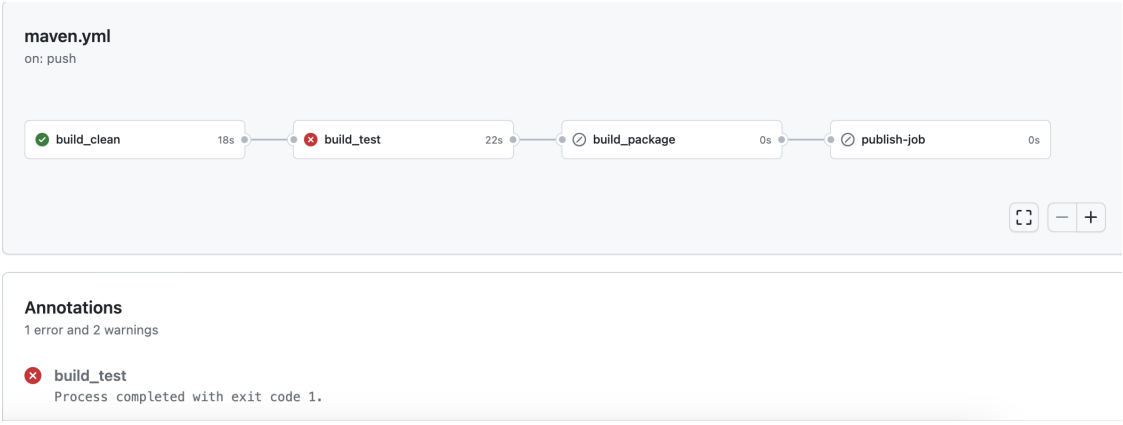

Figure 2: CI/CD pipeline failing at Build and Test Phase

4.3 Scenario 3 - Build Fails

This a walk-through of an unsuccessful push to the CI/CD pipeline, with an explanation of why the push failed, and the repository wasn't deployed to a docker image.

- Code is pushed to `GitHub`, and then the project is built. In this case, there are compile-time errors, and the push is rejected before reaching any other stage.

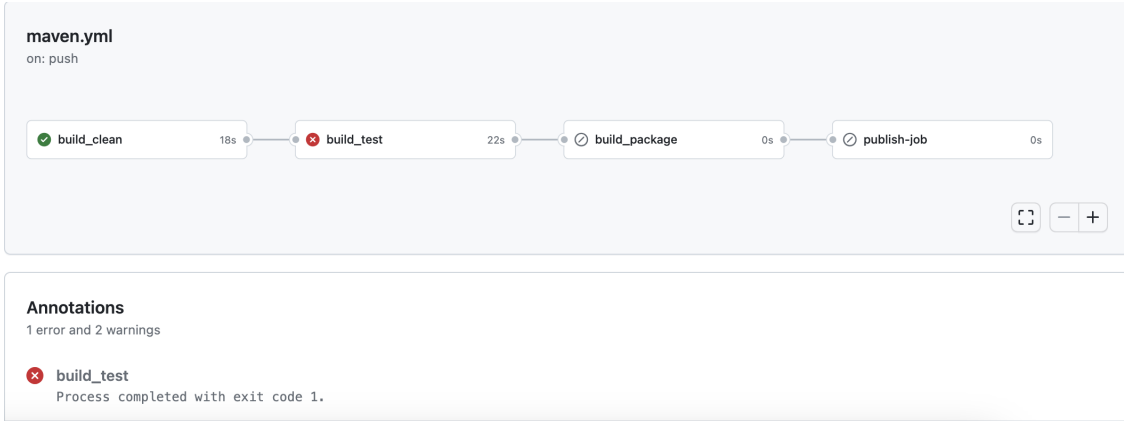

Figure 3: CI/CD pipeline failing at Build and Test Phase