UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Continuous Integration Report
Cohort 2 - Group 16 (Skloch)

Group Members:

Charlotte MacDonald
Hollie Shackley
Luis Benito
Kaustav Das
Sam Hartley
Owen Gilmore

a)
CI Methods and Approaches

Our project employs a straightforward but effective Continuous Integration (CI) pipeline suited for a small team of developers using GitLab (because of its easy to learn UX) [2]. We implemented a single pipeline in our CI process to manage the build, test, and release phases efficiently.

Its usefulness shows in the way it simplifies coordination amongst our team members and streamlines our development process, ensuring all contributions are integrated and tested and errors are quickly identified and resolved. [3]

Pipeline Details

Overview: The pipeline is triggered by any new commit pushed to our main branch in the version control system (for every push to the repository). This ensures that every change undergoes testing before integration, maintaining the stability of our project.
Build Job

Input: Source code from the repository.
Output: Compiled program or an executable.
Process: Converts source code into a runnable state, verifying that the code is free of compilation errors and passes unit tests.

Release Job

Input: Successfully tested build.
Output: Packaged application ready for release.
Process: Tags the current build with a release identifier and prepares it for deployment. This step occurs only when a merge into the release branch is detected, facilitating controlled and scheduled releases. Will include the executable files for each operating system.

Benefits for the Project

The CI setup brings several benefits to our project and our team narrowed down on the key benefits for the sake of brevity. These are:
- **Immediate Integration:** Changes are immediately tested and integrated, reducing the integration issues that are more commonly found in the later stages of inexperienced team-based projects (that don't implement CI pipelining.)
- **Enables Refactoring:** Frequent and immediate integrations mitigate the risks associated with long-term refactoring efforts, like difficult merges and potential for introducing bugs (which allows for immediate feedback and adjustment, reducing the overhead typically associated with refactoring). And thus, the codebase remains clean and adaptable. [1]
- **Automated Testing:** Ensures that all unit tests must pass before they are integrated, thereby maintaining code quality.
- **Awareness/Transparency:** Provides visibility into the state of the application and health of the project at any time by all team members.
- **Frequent Releases:** Simplifies the release process, allowing the team to deliver updates regularly and efficiently. (regular updates can be done easily by just updating the tag).

Conclusion:
Our CI pipeline is tailored to the needs of our small team of 6 people, focusing particularly on quick integration, testing, and release cycles. This approach not only improves product quality and team productivity but also aligns with best practices in software engineering, making it an ideal choice for our project.

b)
<u>Implementation of Continuous Integration Architecture</u>

Our project's CI pipeline was set up in GitHub Actions which uses YAML syntax to define the workflow [4]. We chose to use this as our project is already in a GitHub Repository for version control. It integrates seamlessly with our project and can be triggered by pushes and pull requests.
The pipeline is set to run when any push is made to the main branch. This includes when the tag has been updated with a new version number. It will also run on a pull request or merge to the main branch. Our implementation is divided into 3 jobs: build, dependency-submission and release.

Build

This job is set to run on 3 different operating systems using a matrix: ubuntu-22.04, windows-2022 and macos-12. Permissions for this job have been set so it can read contents. It uses actions/checkout@v4 which allows the workflow to access your repository [5]. Then there are 5 steps:

- Set up JDK 11 - Uses actions/setup-java@v4 to download and setup a version of java, requested by setting the java-version parameter to '11', as we are working with Java 11 [6]. It also specifies the distribution which is Eclipse Temurin, a Java Standard Edition build [7].
- Setup Gradle - The gradle/actions/setup-gradle action sets up the Gradle environment which allows it to cache state between workflow runs and provide a summary of all Gradle executions [8]. As we are using Gradle 8.5 the gradle-version is set to '8.5'.
- Build with Gradle 8.5 - This is responsible for building the Gradle project after it has been set up.
- Upload library JAR - Uses the actions/upload-artifact action which will upload the JAR file as an artefact [9]. This uses the name and path of the JAR file in order to find it in the project to upload.
- Upload Checkstyle report - Also uses actions/upload-artifact. Will upload the checkstyle report when given the path to where the report will be. The report uses a Checkstyle plugin for quality checks on the project's Java source files [10].

Dependency Submission

This job is set to run on the latest version of Ubuntu and has permission to write to the contents. There are 2 steps:

- Set up JDK 11 - The same as in the Build job.
- Generate and submit Dependency Graph - Uses the action actions/dependency-submission to generate a dependency graph for the project and submit it to the repository, allowing GitHub to alert about reported vulnerabilities in the project's dependencies [11].

Release

This job is set to run on ubuntu-22.04 and can only run after the build job has been successful. It will also only run if there has been an updated tag in the push/pull request. Permissions are set so this job can write to the contents. There are 2 steps:

- Download Ubuntu Build Artefact - actions/download-artifact [12] is used to download the JAR file that was uploaded in the Build job.
- Release - Uses the softprops/action-gh-release action to create a GitHub release when the tags are updated [13]. Includes the downloaded JAR file in the release.

References:

Article

[1] M. Fowler, "https://martinfowler.com/articles/continuousIntegration.html", Benefits of Continuous Integration, January 18, 2024. [Accessed: April 20, 2024].

Abstract Article

[2] M. Rehkopf, "https://www.atlassian.com/continuous-delivery/continuous-integration", Continuous Integration Tools Comparison. [Accessed: April 24, 2024].

Textbook

[3] H. Wright, T. Winters, and T. Manshreck, "Software Engineering at Google: Lessons Learned from Programming Over Time", Ch.23:Continuous Integration (written by Rachel Tannenbaum) and Ch.24: Continuous Delivery (written by Radha Narayan, Bobbi Jones, Sheri Shipe, and David Owens), 1st ed. California, O'Reilly Media, Inc, 2020. [Accessed: April 24, 2024].

Documentation

[4] GitHub Actions, "Understanding GitHub Actions" GitHub, https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions [Accessed: April 30, 2024].

Source Code

[5] GitHub Actions. actions/checkout [Source code] https://github.com/actions/checkout [Accessed: April 30, 2024].

[6] GitHub Actions. actions/setup-java [Source code] https://github.com/actions/setup-java [Accessed: April 30, 2024].

Documentation

[7] Adoptium, "Eclipse Temurin™ Latest Releases". https://adoptium.net/en-GB/temurin/releases/ [Accessed: April 30, 2024].

[8] GitHub Actions, "Building and testing Java with Gradle" GitHub. https://docs.github.com/en/actions/automating-builds-and-tests/building-and-testing-java-with-gradle [Accessed: April 30, 2024].

Source Code

[9] GitHub Actions. actions/upload-artifact [Source code]. https://github.com/actions/upload-artifact [Accessed: April 30, 2024].

Documentation

[10] https://docs.gradle.org/current/userguide/checkstyle_plugin.html [Accessed: April 30, 2024].

Source Code

[11] GitHub Actions. actions/dependency-submission [Source code]. https://github.com/gradle/actions/blob/main/dependency-submission/README.md [Accessed: May 1, 2024].

[12] GitHub Actions. actions/download-artifact [Source code]. https://github.com/actions/download-artifact [Accessed: May 1, 2024].

[13] D. Tangren (softprops). softprops/action-gh-release [Source code]. https://github.com/softprops/action-gh-release [Accessed: May 1, 2024].