UNIVERSITY OF YORK

DEPARTMENT OF COMPUTER SCIENCE

Continuous Integration

Cohort 2 - Group 17 (Rich Tea-m 17)

Group Members:

George Jopson
Ben Slater
Meg Tierney
William Potts
Jamie Burgess
Seyi Towolawi
Alex Staicu

Methods and Approaches

Given the risk of integration related errors that come with a project in which multiple features of a project must be implemented in parallel and must function together without conflict in the same software build, we have employed several tools and agreed upon principles as a team to minimise this risk and ensure the development process is smooth.

The idea of continuous integration stipulates that components of a project must be tightly integrated as they are developed through simple iteration. To follow with this and thereby minimise integration related errors, we have laid out the following principles which we will follow throughout our work on the project, inspired by the version control management practices of Trunk Based Development:

1. We will make efforts to conform to the Trunk Based Development Branching Model. This is a model for using branches in a version control system such as Git, in which the team uses as few branches as possible in the repository [1]. Given the numerous features which must be implemented in this project, branches can be a useful tool to allow one component to be iteratively worked on without impeding another component. However this comes with the risk that upon merging these branches, we will find integration related errors across components, so to minimise this possibility we have agreed to use them within a limited scope and merge them back into the main branch as soon as this scope has been reached. This model allows for extremely tight integration, and although it requires a great deal of coordination from the team, this disadvantage is manageable given the limited scope of the project.
2. When a team member is implementing a project feature, they will aim to implement that component in its simplest form before ensuring its integration with all other components. This conforms with the practices of Trunk Based Development, in which updates must be small and frequent updates must be regularly added and integrated with the main “trunk” branch of the project [2]. In a project in which all components are interacting with each other constantly, there are many potential points of failure from which errors will emerge, and a slow, iterative process is the best way to combat this, allowing integration errors to be identified quickly after they emerge and be eliminated.
3. All builds will be automatically tested to ensure they are integrated with other system components, and when tests are failed, priority will be given to fix whatever issues have emerged. This is the most important step to ensure continuous integration is upheld [2]. The previous principles minimise the chance of errors, but once they do emerge they must urgently be dealt with to minimise the long term issues that come from integration errors, where components are completed but their lack of integration leads to enormous amounts of wasted time. By automating this process, the engineering team will easily spot errors without time consuming manual testing.

The method we have chosen to facilitate the application of our principles is to create a gradle.yml file in the project and use features of GitHub Actions. There are two reasons for this choice:

1. Since we have already been using a GitHub repository to manage our project across multiple devices, GitHub Actions was the simplest way to incorporate integration tools into our workflow.
2. Since these tools were meant to make frequently repeated actions more efficient, it was important that we ensure the tools are convenient to use instead of creating fuss, so the accessibility of GitHub Actions made it by far the best choice.

Implemented Infrastructure

There are several functions the gradle.yml file has. For every commit, a build is created in GitHub with two artefacts, a JAR file of the build and a JaCoCo test report. Creating the build of course confirms that the project is a functioning piece of software and facilitates the provision of the artefacts, each of which allow for testing. The JaCoCo report uses tests we have created and included within the project to ensure requirements are met and the JAR file allows the committed version project to be run and tested directly. These artefacts improve the efficiency of the third principle, making it quick and easy to ensure standards of integration are maintained.

We have also included the option to release a new version of the project with a JAR file easily by pushing a tag in a commit labelling the version of the project. This allows us to use the continuous deployment software release process [3], where if tests are passed we can easily release a new version to rapidly collect user feedback, increasing the speed with which the team can respond to their needs. However this is less essential than ensuring the project remains continuously integrated.

To summarise, the gradle.yml file does the following to every commit that is pushed to GitHub:

1. Creates a build of the project in GitHub Actions, confirming that the build is working without any errors.
2. Adds a JaCoCo test report as an artefact to the build, which can be viewed to confirm the build passes tests we have included in the project.
3. Adds a JAR file of the build as an artefact in the build, which can be downloaded and run, allowing users to test that version of the project.
4. If a tag is included in the commit labelling it as a version (e.g. v1.0.3), a new release is automatically created on GitHub, including a JAR file of the version, allowing anyone to download and test or use it.

References

[1] VamshiK, “CI/CD: Github/GitLab Branching Strategies”, Medium, Oct. 28, 2023. [Online] Available: https://medium.com/@katla.vamshi/ci-cd-branches-strategies-449befdeb1b5 (accessed May 14, 2024).

[2] “Trunk-based Development,” Atlassian. [Online] Available: https://www.atlassian.com/continuous-delivery/continuous-integration/trunk-based-development

[3] “Continuous deployment,” Atlassian. [Online] Available: https://www.atlassian.com/continuous-delivery/software-testing/continuous-deployment