CI Report

Part A - Methods, Approaches, and Procedures

The main aim of practicing continuous integration is to increase the development pace, and respond to new changes to the codebase rapidly. Our team's development process consists of a mixture of FDD and RAD. With multiple CI-related practices, our team was able to make sure that the development pace is fast, and the code produced is of high quality.
We have introduced both new tools and development process changes, as well as guidelines that team members follow.

By using a VCS and a code hosting facility, our team is able to propagate and integrate new changes to each team member quickly.

Code quality

Team members may have different stylistic approaches to code, as well as different conventions and idioms to follow. To ensure the code quality is consistent and high, we have introduced a standard code style, and incorporated code linting. Members are informed of the code formatting errors by the linter.
Developers may unintentionally introduce bugs and common mistakes into the codebase. Our team has integrated static analysis tools into our build system. This allows us to catch common mistakes and fix them before actual integration.
With these facilities in place, team members are working on a high quality codebase that is visually consistent, and relatively bug free.

Deployment

Instead of relying on team members on building the artifacts on the developing machines, deploy, and distribute it, the CI automaton builds in a standardised environment and releases the artifacts with each iteration of integration.
This enables us to create consistent, reproducible builds, and distribute it (through the CI artifact archive) in a timely manner. The automation also enables continuous delivery, where artifacts are effectively snapshots of each integration.
Developers need not to intervene in the build and deploy process at all times, which means that we can work more efficiently and allocate our time on the software development itself.

Badges

To give a clear indication of the status of the codebase, in terms of CI practices such as passing tests, and successful builds, badges are added to the codebase, in the readme file, and they are updated automatically with each iteration of integration. This gives a clear indicator to the developers (and people who came across the repository) how the latest integration went.

With all these in mind, we have created an integration machine that tries to build the source tree when team members create pushes, catch bugs and errors, and notify about the success or failure. We have also automated the creation of badges, and integrated our project build tool with the CI system.

(Code coverage measurements are not employed in our codebase as we did not implement white-box testing, as mentioned in the Software Testing document.)

Part B - Actual supporting infrastructure

Our team uses Git extensively for version and branch control, with GitHub as our code hosting facility. We have utilised GitHub's automation system, GitHub Action, to facilitate our CI practices.

With respect to the codebase and build system, we have added

- Checkstyle for code linting, and
- SpotBugs for static analysis.

The CI automation system also supplies automated badge creation based on the status of the CI workflow, such as build success/failure.

The automaton also builds a JAR distributable using the build tool. It fetches the built JAR, and creates a release with the current revision (version) number.

Our CI automation system is also integrated into our team's communication channel. A dedicated channel is created in Discord, and webhooks are set up in GitHub. All notifications related to the project, including Git push to branch, artifact releases, and CI automaton status, are sent to the channel.