Continuous integration

Cohort 1, Group 5
Team Name:
JAzZ MoLeS

Group Members:
Sophia Taylor, Lucy Wood, Mitchell Gilbert
Jamie Creed, Archie Adams, Zayed Iqbal

For this project, we used a software development practice called continuous integration which by martin fowlers [1] definition is where members of a team integrate their work frequently, with each integration being verified by an automated build (including test) to detect integration errors as quickly as possible. We felt this was vital to the success of this project since it improves our software quality, and we can all address bugs at a quicker rate than we would without it.

The main methodology and approach we followed was maintaining a single source repository using git. Maintaining a single source repository showed itself to be extremely useful. It saved our team members from any confusion of where things were, as the majority of the sources we deemed were necessary for this project were located in the main branch, with any branches made being short lived as to avoid any difficulty with re-integrating that fork back into the main build/branch.

We chose Git specifically for code management for a variety of reasons. Software developers tend to prefer git over other version control systems like CVS and Perforce, since Git has the adaptability, speed and stability required to thrive in fast paced markets. Additionally to this, the majority of us have some experience with this tool - which in itself is beneficial, as with this experience, we as a team can identify any potential problems earlier and develop more effective solutions with a higher level of efficiency.

Other approaches that we implemented within git were creating a series of automated tests, and everyone constantly keeping up to date with the commits being made.These tests are Junit tests and they are run by a github actions workflow using gradle. The advantages of these are: with automated testing, it prevents you from releasing any buggy builds to users by mistake, as well as acting as a safety net making it easier to identify any issues. These issues are then fixed immediately, as kent beck says “nobody has a higher priority task than fixing the build” [1]. We felt it was integral that we kept this in mind to maintain the quality of the code, rather than leaving it and it becoming a big bug that's more difficult to sort.

Additionally, in terms of making sure that everyone is familiar with the commits being made, this is so that it should be a lot easier to see what state the project is in and how the various builds are going. This helps towards creating deadlines for testing, user evaluations and eventually deadlines before the last submission of this project.

To summarise the continuous integration infrastructure we used: we set up Github Actions to execute the Gradle build, as well as execute automated tests. This way, with every commit made to the Github repository, the team member receives instant feedback; if the build was successful and all specified tests were passed, the commit is labelled with a green tick. If one or more of the Github Actions fails, the commit is labelled with a red cross: this indicates a bad commit. Github also provides easy-access details of each check, so that the commit author can easily see the source of any issues.

We used the JaCoCo plugin to generate coverage reports. Code coverage is a metric used to measure how much code is executed during automated tests [2]. If part of the code is missed, this could mean bugs are being missed. Therefore it is optimal to aim for a high coverage level when running a set of tests. JaCoCo reports help to visualise this for our team members, allowing us to see roughly how effective our test coverage is.

Additionally, we used the Gradle plugin Checkstyle to ensure all code met Google style rules.This is done automatically as part of the Github Actions process, and it means that every commit, any issues with the code style is immediately flagged, allowing it to be immediately addressed as per the continuous integration approach.

Reference list:

[1] M.Fowler. Continuous Integration [Online]. Available at: https://martinfowler.com/articles/continuousIntegration.html#PracticesOfContinuousIntegration [Accessed: 15 May 2024]

[2] S. Pittet. What is Code Coverage? [Online]. Available at: https://www.atlassian.com/continuous-delivery/software-testing/code-coverage [Accessed: 15 May 2024]