Continuous Integration Report:

Team 14

Joshua Beaven (jb2825@york.ac.uk)

Calum Feenan (cf1218@york.ac.uk)

Harry Hillman (hth512@york.ac.uk)

Jonathon Snelgrove (js3336@york.ac.uk)

Omar Alhosani (oaa560@york.ac.uk)

Rayan Butt (rb1770@york.ac.uk)

Continuous Integration Report

Part A)

For our project, we thought it was vital to implement a continuous integration (CI) method that automated the action of building code, integrating it into the codebase, and testing it on a regular basis. The code was then updated in the Github repository. The reason behind this was we knew there would be fewer bugs and errors overall when adding code, a faster program production time, and a decomposed, manageable workload. We therefore set up a CI/CD pipeline within Github to go about doing this.

Overall, we were happy to go about with this approach as although our planning method, SCRUM, doesn't require a CI pipeline, and it is a methodology that emphasises the importance of frequently updated and incremented codebases, which was exactly what we planned to do. Using our CI pipeline allows our team to communicate our code better, update our code more, allow for less error merging, and therefore speed up the development process by adding little changes.

In order to achieve this, we identified four important features that should be implemented in our CI process:

Version Control System (VCS): The code and all necessary artefacts for building should be stored in a VCS. The VCS must be capable of identifying code conflicts during integration and tracking code changes. CI must be able to build the latest version of the code.

Automated Build: The build process should run automatically without human intervention. Users should not need to type commands or click buttons.

Testing: The build must run both automatic and manual tests, including unit and integration testing. Discussion is needed about what percentage of code coverage is sufficient. Test results should be available after CI is complete to trace any issues that occur.

Separate Integration Machine: The build should run on a separate integration machine to ensure that everything can run elsewhere, not just on the local machine it was developed on.

By implementing these key features in our CI process, we can ensure that our code was continuously integrated, frequently updated, and, most importantly, has no errors.

The main tool we're choosing to use for continuous integration is GitHub.GitHub is a repository hosting service for Git that also has a web-based graphical interface. It is used for open-source projects, which makes it ideal for group projects, as everyone involved can access the project at any time. It also allows for a simple editing and upload process, which is vital in group projects, and edits are constantly taking place. It is also extremely well documented and very easy to learn, as there is a plethora of information on how to use GitHub and what are the best ways to approach any problem you may encounter. Another great feature is that it keeps track of changes made in code across versions, so if there is ever a falter in communications between members of the project team, it is documented when a change is made and which member of the team made the change. It also stores previous versions so that if the updated code ends up causing problems, you can always revert to an older version of the code.

Another tool we will be using is Gradle which can be used in conjunction with GitHub. Gradle is a build automation tool for multi-language software development. It controls the development process in the tasks of compilation and packaging to testing, deployment, and publishing. This can be used in conjunction with GitHub and is a very effective tool when it comes to group development of projects, as it has a multitude of tools to maintain consistency between code in projects and makes uploading, downloading, and editing code easy for everyone involved. Gradle has allowed us to use other tools within Gradle such as allowing us

Part B)

From reviewing our planning, we decided to continue with the method of using the four features in our CI pipeline. On further research, we discovered that Github provides a continuous integration feature called ‘Github Actions’. This was a feature that none of our group had used before, so were unaware of it until learning more about the CI process. We thought it would be best to learn and use Github Actions due to the fact that we had used Github for our previous assessment and were familiar with the procedure of using repositories, branching code, and more. We also liked this idea to continue with Github as our accounts on IntelliJ & VScode were connected with Github, so the transition to using Github actions was rather intuitive.

Our system for CI worked as follows:

- A new branch was made from the code, for example, ‘JoshTestingBranch’, or ‘Harry_Reputation’, to create a new branch of code signifying what each person was working on. This code was forked from the ‘main’ branch, which is the branch of code that everyone pushes to update the codebase, with all dependencies installed and built on. This was a necessity needed for our integration.
- The prewritten unit tests / manual tests were then automatically run on the code to highlight any bugs or errors that we may have missed or created accidentally. This was key for making sure that we didn’t push any code with errors before merging them with the main branch, which would slow the development process further.
- If the code passed each test, it would then go to the staging environment automatically for further tests and see if the code would work in live production. This would be to make sure that the code works fully with the rest of our project.
- Finally, the code is then pushed onto the main branch, allowing for the update of development.

We used this process and software as it had many visible benefits:

- Each member of the project could fork the code as and when needed, allowing anyone to work on the code at any moment in time, no matter the time or place, and without the need for interaction from others. This was beneficial as each member could work on a different part of the project without interfering with anyone else. For example, testing and implementation.
- We could also see when other team members ‘committed’ their code to the main branch, allowing us to be notified when the main branch was updated and therefore keeping us all up to date with the updates of the project. This was beneficial as rather than taking longer times to update the code, we all knew the exact status of the project at any given time and could follow the progress, meaning we could know what to do next quickly.
- Our own software, ‘IntelliJ’ or ‘VScode’, could seamlessly update to the latest branch of code when possible so that no conflicts could arise; no one developer’s branch could get too outdated to no longer be as effective or compatible. Overall, we believe our approach did work effectively, and we executed it well. We chose Github for our VCS, which was an effective choice as we were all familiar with the software, and it worked well, allowing all of us to work on the project simultaneously. For automated builds, we found that we could choose our own parameters for when tests were made and code was built. For example, our pipeline would trigger each time a member wanted to push to the main branch. This would make sure that the code is ready before updating the project. For testing, we found that GitHub worked well as every member of the team had the latest version of the codebase, meaning tests were effective and not redundant. We were also highlighted by Github when we made changes to the code, and any code did not pass the automated tests set out in ‘Github Actions’. Finally, for a separate integration machine, Github would test in the staging environment before live production to make sure that code works. Additionally, the use of GitHub allowed us to download the project on multiple devices when connected to the internet, allowing us to check for success on other systems easily.

Appendix:

https://about.gitlab.com/topics/ci-cd/
https://semaphoreci.com/blog/cicd-pipeline