| | |
|---|---|
| **Module** | ENG1 |
| **Year** | 20/21 |
| **Team** | 33 (Short Circuits) |
| **Members** | Jack Lord, Neo Metcalfe, Sam Rodgers, Mohammad Abdullah, Qi Tang |
| **Deliverable** | Continuous Integration Report |

A)

In our approach to the project, we have used Github to store code in a central repository which all members can access and edit. This allows all changes to be centrally controlled and can assure that all members are running the same version of the project. If they didn't pull a recent commit and then pushed their changes, then any new changes from other members of the team would be removed.

At the start, there were a few issues with members of the team not pulling up- to-date versions of the code. This resulted in other members of the team having their work overwritten. In order to fix this, we decided to make separate branches which can then be loaded onto the newest version of the main branch. These separate branches show how many iterations in front or behind it is from the main branch. Therefore, this allows each member to work on their own part of the code and see if it interacts with the main branch correctly. Then, when this is confirmed they can upload it to the main branch.

However, we then found that members were at various levels in front or behind of the main repository, and then therefore reverted to having a single branch. We then reiterated making sure that everyone pulled at frequent intervals to make sure that their code was up to date. This single repository setup ended up solving these issues and the reinforcement of making each other pull frequently meant we didn't have too many future issues.

The weekly meetings allowed us to report what work was being done so that we each had an idea who was doing what and how it tied into our own work. This meant that if certain parts didn't integrate well, then we knew who to discuss it with.

Our CI included an automated build and testing feature which saved a lot of trouble and time.

Whenever a build was broken or an error was thrown, we decided to tackle the error with priority. This meant that no code was then committed in future before that error was resolved. This helped to reduce the amount of bugs.

B)

The CI we used was GitHub Actions working through Gradle. This allows all pushes to be built and run against tests to make sure that all changes work.Github Actions looks at the language and framework that you are using and recommends the continuous integration workflows that would work well. Gradle is a build tool which works for Java and can be run through Github Actions. This is used to automatically build the tests for Github.

Using the automation tool allowed us to push code and see if it works without having to manually check each time which saved us a lot of time in the process. This also reduces the amount of errors that each push could have caused if a test was run incorrectly.

Test reports are also automatically published which means we could view each test individually to see (if anything) was wrong and methodically deduce what is wrong. This, again, saves a lot of time.