Version Control

# Google Docs

In regards to version control, Google Docs will be used to keep track of any documents we write. Documents such as our ‘meeting-minutes’ will be stored on Google Drive. Google Docs allows the three of us to have access to whatever has been said in meetings, without having to rely on one person to have the document at all time; this also succeeds in eliminating the concern that members of the team may not be aware of what has been said in meetings, or that they may have forgotten what has been said. Having it on a cloud based system also means that in the incident a member of the team may no longer be a part of the team, any new member can easily access these documents and be caught up to date with what has been discussed in prior meetings. The version control aspect of Google Docs allows us to either revert to an old version of a document we have written: for example, in the case something we have written is no longer valid; or to note contributions in a document.

# GitLab & Eclipse

For software development, we will be using GitLab. A shared GitLab project will allow each member of the team to access the code and make their own changes to it. We will be using eGit: a tool used with the Eclipse IDE to work with Git Repositories. By cloning the source repository, each member of the team can work with the whole project in their own workspaces. Every member of the team will be able to access the complete version of the project, and will be able to pull from the source repo, to their own local repository. The main benefits in version control with Git, are: the comparison of different files – letting us for example, compare a file with bugs and a file prior to that file that may not contain any bugs, to help ascertain where said bugs came from. When pushing code to the remote branch, we will use merge requests. This will allow us to review code before merging into master. Each developer will push onto separate branches, meaning they can work on their own features without it interrupting the rest of the project. When those features are implemented, they can be merged.

Process for using GitLab with project is as follows:

1. Each developer will create their own branch of the project to work in.
2. Tasks assigned to developers in meetings will be completed in developer’s own branch.
3. Developer will clone the repository and have a local repository where they can write code.
4. When tasks have been completed by the developer, the code will be given to a reviewer.
5. The reviewer will review the code based on the reviewing standard from the review document.
6. If code passes code review, feature will be merged into master branch.

Code review is essentially examining the source code in order to:

* Improve the codebase
* Find bugs
* Give advice to developers on their approaches towards programming
* Making sure code stays consistent in its readability and making sure it’s easy to review.

By having other members of the team observe code, it means things that may not initially be observed by said author of the code, may be noticed by other members of the team. This ensures our work reaches and maintains a high quality of work.

Our review process will work as such:

1. When a merge request has been made, the other members of the team will review said code.
2. When noting corrections to be made, reviewers should explain the reasoning behind the corrections, so as to ensure the developer understands what exactly their mistake was. If possible provide documentation or general evidence to back up the reason for the comment.
3. Reviewers should explain why they’re asking certain questions. Rather than just simply asking something like “Why did you not do x here”, it presents an implication that a simple solution could have been used, and fails to actually explain why a different solution may be better. This will not only improve the codebase, but also the programmer’s general skill.
4. Reviewers should only review fewer than 400 lines of code at a time – or less than 500 per hour. [[1]](#footnote-0)Based on research by a SmartBear study of a Cisco programming team, it was discovered developers lose the ability to effectively process so much information at time – with the ability in detecting defects, diminishing beyond 400 lines of code.
5. Code should not be reviewed for more than 60 minutes at a time. People participating in any sort of activity where they must use large levels of concentration will find their performance in said task will eventually begin to start dropping off after around an hour of work.
6. Reviewers should take breaks between periods of review in order to ensure quality of review stays up to a high degree.
7. When code has been accurately reviewed by other members of the team, said reviewers can decide on whether the merge request should be accepted.

1. https://smartbear.com/learn/code-review/best-practices-for-peer-code-review/ [↑](#footnote-ref-0)