**GROUP PROJECT CO600 | Version Control Checklist**

We are very aware that working on a single codebase with more than one developer can be problematic and can lead to loss of work and a whole number of conflicts (merge related and group related) and because of this we have developed some of our own best practices that we all agree on and will be following when making changes to our GitHub repository which is detailed in the points below:

* **Use the branches!**- Branching is one of the most powerful features of GitHub and we have agreed to use this feature to our advantage whenever we work on open issues to avoid merge conflicts.  
  - Specifically we will be branching
* **Commit often, but only work that is complete and tested!**- It is important when using a VCS to commit as regularly as possible. Committing often makes sure that your commits have only related files, and that your commits are small and easy to integrate and makes merges much simpler.
* **Write descriptive commit messages**- Commit messages should explain what the change does exactly in contrast to the previous version and why the change is necessary. The commit message should also start with a verb like 'changes', 'adds', etc...
* **Merge as a group**- All branch merges to the master branch should happen during or directly after a group meeting so that any conflicts can be discussed, and no files are lost.

Using these general rules, we have come up with our own checklist that we need to check off before doing anything related to version control i.e. branching a new issue, pushing to the branch, merging the branch to master. The reason why we have created the checklist is so that everyone knows exactly how to add new features to the project

* Untested code must not be committed.
* Code that doesn't compile must not be committed.
* Committed messages must be meaningful:
  + Messages should not be longer than 100 characters.
  + The messages must answer:
    - What was the reason for the change?
    - How does it differ from the previous version?
    - Use words such as *‘change’*, not ‘*changes*’ to keep consistent with git command generated messages.
* Work must be committed as often as possible.
* Main branches:

○ Master branch:

Latest code development and updates states must be committed to the main repository, all operations related to branching and merging must be from master.

● Changes should be made through the following supporting branches:

* Feature branch - must be used to push new system feature or functionality.
* Bug branch - must be created when a bug is detected, which will be fixed and merged.
* Hotfix branch - is used for performing immediate repair on the project.

● Supporting branches must be closed as soon the task is completed. Putting it all together:

Initially when a certain system functionality is created, a corresponding feature branch is also created. The feature being developed will be pushed into the relevant branch until development process is completed, which then will be merged with the master branch. When a bug is detected in the source code, a bug branch is created to deal with the issue.

With regards to documentation, such as meeting minutes, review meetings, templates and planning documents, each have an individual branch explicitly assigned for these documents. This allows better file management, provides an easy way to interact with branches, quick accessibility to project resources and the ability for clients to easily locate documents and keep up to date with latest changes.