**VCS use report**

Git and Github were our choice for Globex because of the distributed version control and because most developers were familiar with it.

Summary - Before I started to make changes, I wrote tests that currently fail but will pass once I have fixed the bug or implemented the feature. I forked the project to make my own copy of the repository on GitHub. I then cloned it on my local machine. I created a git remote to refer to where I forked my copy from. I ran tests on my code, within my copy, in accordance with the software manual and README file. Then I kept committing logical chunks of work as I progressed. Before the code review, I took care of the following: 1) Testing: I had written tests for the Musoplan features and bug fixes. 2)Documentation: I have documented each procedure that I added or modified, and I have updated the user manual. 3)Completeness - To see if there were issues in other similar code that needed fixing.

Complying with organizational requirements - I made sure of :

* Using Git which was preferred at Globex
* Starting the Musoplan repository blank and configuring settings so that commits reflect my name and address
* Using ‘main’ for initialisation and branches for Musoplan development
* Used commit messages that followed industry standard, were frequent and had a scope of one change at a time

Expected VCS behaviours -

When ready to start on a unit of work, such as fixing a bug or implementing a feature, create a new branch. Pull upstream from time to time to incorporate any work done by others since you created your branch. Add files, modify them, stage them by pushing to Git and then commit them. Resolve merge conflicts by checking that all tests pass on the fork locally, push the changes to remote and check if the fork still passes all tests there.

When wanting the work to be incorporated into the project's main repository, create a pull request for other maintainers to look at it. Make a descriptive title for the pull request that can later become the commit message. The maintainer will resent back a review. Respond to the feedback by making changes in the working copy, committing them, and pushing them to GitHub when the tests pass locally. The pull request will get automatically updated. Request a re-review and wait for changes to be accepted as part of the project. Delete the branch, after it has served its purpose.

As a repository owner/maintainer - To keep the version control history clean, select “Squash and merge” when you merge a pull request. “Squash and merge” results in a single commit that contains all the changes in the pull request.

References –

Ernst, M., 2021. *How to create and review a GitHub pull request*. [online] Homes.cs.washington.edu. Available at: <https://homes.cs.washington.edu/~mernst/advice/github-pull-request.html> [Accessed 31 August 2021].