# Managing and storing our methodology programs

Meeting date: 04/03/2019

Notes by Ella, 05/03/2019

Present: MRCCTU: Ian, Tim, David, Kevin, Ella; central IT: David Perez-Suarez (DPS)

Apologies: Andrew

**Aims of meeting**

With the increased focus on software development in the MRC Unit, there is a need to formalize our software testing and standardize the process of our software development. We will predominantly be focussing on the development and dissemination of Stata programs.

**Testing**

DPS explained general testing techniques, in particular:

*Unit testing*. Checks CORRECTNESS, i.e. checks that the code provides the expected result.

*Regression testing.* Checks STABILITY. Used when existing software is refined and the new improved software needs to be checked to ensure it still provides the same result as the original code.

The Stata users in the group explained that when a new Stata program is emailed to ssc, it is uploaded without any external checking/testing – it is the responsibility of the program owner to fully test their program. The Stata Journal provides the opportunity for users to publish details of their new program through a peer reviewed system.

If a new version of the program is emailed to ssc then this will overwrite the existing version entirely. It is not possible to perform ‘regression testing’ as there will be no previous versions available externally, other than those already downloaded previously by users.

**Version control**

As ssc does not provide a utility for storing previous versions of code, it is thought that Github could be used as a way of implementing version control and providing access to previous versions, as well as the testing files and documentation of our programs.

DPS commented that it would be useful if Github users could read and re-run the testing files themselves if they so wished.

**Publishing software**

This is our main goal.

The issue of licensing was raised, and the fact that the MRC might have a policy on which licence is used as well as any copyright issues.

DPS recommended MIT and BSD licencing, there is also the option of GPL. He also advised that different projects can have different licences, but if the license changes in the future then all contributors have to provide their permission. UCL will be publishing their position about open source software licensing soon.

**Action: Tim** offered to find out which licence the MRC would like us to use.

In terms of linking to Stata, the Stata program help file could include a line which points to the Github directory.

DPS also suggested including a public log of issues on Github and a change request form, whereby users can make requests for additions/improvements to be made.

**Github**

Multiple users can work on the same project in Github by using branches. There will be one main branch of the project, the ‘stable’ branch, which is released publically (and will contain the associated program published on ssc). The stable branch will not change until the next version of the program is ready for public release.

The other branch(es) are for development. Once the user is happy to release their updates, they can merge in to the stable branch and release their program.

DPS advised that there should only be one designated person per project who has the authority to accept changes to the programs (through Github ‘pull request’), and has the overall responsibility for the organization of the directory (the ‘directory owner’). This protects from people accidentally pushing programs in to the stable branch, as the main person will first have to approve the request before the branch is merged. My follow up question: what happens if that person is no longer available to work on Github? Can this authority be changed at a later date?

Note that a Github FORK is a clone on Github alone, a Github CLONE is a clone on your machine.

Anyone without permissions for the directory (e.g. someone external to the Unit) can create a fork, make their changes and create a pull request to invite the directory owner to view and accept/reject the changes.

Forks can also be deleted at a future date, for example if a collaborator leaves the group.

If the directory is public, any Github user can create a fork.

If the directory is private, forks of collaborators will be deleted if they leave the group.

The power of Github is that is allows a distributed network of multiple users able to fork and create multiple pull requests, however it can be a challenge to manage.

**Creating a new project**

It was suggested that with our group’s new software, we keep the project private with one dedicated owner to accept changes to the master/stable branch. Once the project has reached a certain level of maturity and undergone testing, it can then be made public where every change/commit will be available for the wider Github community to view.

A private Github repository can be made public through the settings tab.

Further software development that needs to be private for a certain time - for example classified research or protecting from other groups publishing first - can be achieved by creating a private fork on the main directory before this is later pushed in to the master branch.

DPS suggested that a change log is included in the main Github repository, where all changes and bug fixes are documented.

Github can also be used to store documentation on processes, agendas and meeting notes.

It is also possible to set up Github so that a group of testing files are run before each commit to the master branch, and to modify the settings so that a branch can not be merged unless the tests have passed. DPS showed the group an example called Travis on Github, where test scripts are run on multiple versions of operating systems and platforms (eg. Mac, windows, linux etc). They use Legion, which also has Stata. This system is called ‘continuous integration’.

It is also possible to write scrips to add/change/remove users’ permissions, which is useful for large projects with multiple collaborators.

DPS also advised on naming conventions for branches - to ensure that it is clear what each branch pertains to and also to ensure that no two people create branches with the same name at the same time. Labelling branch names with the developer’s initials/the issue that the branch is fixing is a recommended naming convention.

**User interface**

Gitkracken is thought to be good for visualisation (like source tree).

Vscode has a plug in for most programs eg. Stata, and has Git integration. It is possible for the user to see what has been changed and push to Github directly.

DPS is providing documents to our group on how best to set directories up, naming conventions etc.

**Additional resources**

DPS directed the group to ‘Slack’ at UCL, which has a couple of useful channels relating to the work we do as well as ‘help me’ sections, where people can have one-to-one and group discussions on any issues they face. A different slack account is required for each team that you join. DPS also mentioned talks/meetings that are held every month and has already provided information on these by email. Ian also mentioned that Will Everett understands Github very well and is willing to help with any questions that might arise.

**Next meeting**

Proposed for our internal group, in a month?