GitHub in the Classroom: Lessons Learnt

Yu-Cheng Tu, Valerio Terragni, Ewan Tempero, Asma Shakil,
Andrew Meads, Nasser Giacaman, Allan Fowler, Kelly Blincoe *
University of Auckland
Auckland, New Zealand
(yu-cheng.tu,v.terragni,e.tempero,asma.shakil,andrew.meads,
n.giacaman,allan.fowler,k.blincoe)@auckland.ac.nz

ABSTRACT

The decision as to whether or not, and how, to use a Version Control System (VCS) in teaching is complex to make. There are a number of use cases for how a VCS can be used in teaching, there are several VCSs, each VCS has a variety of options for how to access them, each has a number of third-party support systems, and all combinations have different benefits, costs, and challenges. At University of Auckland, we have made significant use of Git and related systems (especially GitHub and GitHub Classroom). In this paper, we offer the lessons we have learned from our collective experience. While we by no means cover all of the possibilities, we hope that instructors considering the use of VCSs, in particular Git, will find the lessons we have learned helpful in making their decisions regarding how to use VCS in teaching.

CCS CONCEPTS

• **Applied computing → Education**; • **Social and professional topics**; • **Software and its engineering → Designing software**;

KEYWORDS

GitHub, GitHub Classroom, Version Control Systems, Student Experiences, Code Review, Team Work, Program Assignments

ACM Reference Format:
Yu-Cheng Tu, Valerio Terragni, Ewan Tempero, Asma Shakil,, Andrew Meads, Nasser Giacaman, Allan Fowler, Kelly Blincoe . 2022. GitHub in the Classroom: Lessons Learnt. In *Australasian Computing Education Conference (ACE '22), February 14–18, 2022, Virtual Event, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3511861.3511879

1 INTRODUCTION

As of today, Git[1] is the most popular version control system (VCS) [8]; in the 2021 Stack Overflow survey, over 93% of professional developers reported using Git [28]. As a consequence, we are seeing increasing usage of Git in university courses [8, 12, 18, 23]. In particular, the hosting service GitHub (GH)[2] and its programming education product GitHub Classroom (GHC)[3] are becoming popular among instructors.

The use of VCSs offers a number of benefits to support teaching, such as distribution of teaching material [12] in particular assessment support [26], collaborative development of teaching material [12], and providing experience for teamwork [17]. This has become significantly easier with the development of distributed VCSs such as Git, in particular with cloud-based collaborative support services such as GH, GitLab[4], and Bitbucket[5].

Some cloud-based services now offer free products and support for instructors and students to encourage Git uptake. GitLab has *GitLab for Education*[6]. Bitbucket/Atlassian offers *Bitbucket Education for Student Developers*[7]. GH, through GH Education[8], offers *GHC* and the *GH Student Developer Pack*[9]. At University of Auckland, we have mainly used GH and GHC, and they have primarily been used to support various forms of assessment.

There is a wide variety of possibilities for how Git, GH, and GHC can be used in teaching, with different benefits, challenges, and costs [5, 8]. At University of Auckland, we have experienced many of these possibilities. In this paper, we offer our collective experience, including lessons learned. While we by no means cover all of the possibilities, we hope that instructors that are planning to use (or are currently using) Git in some capacity in their teaching will benefit from our experience.

The rest of this paper is organised as follows. The next Section provides an introduction to the various concepts relating to the systems and technologies we discuss. Following that, Section 3 discusses research on use of VCSs in teaching. Section 4 gives an overview of the courses on which we base our experience, to give an idea of the variety of uses we have made with Git and related technologies. The main contribution is in Section 5, where we present our experience, issues we faced, and approaches we took to resolve them. Finally, Section 6 presents our conclusions.

2 BACKGROUND

This Section provides a brief introduction to GH in the Classroom. While we do not focus exclusively on use of GHC in this paper, in order to understand it one must understand many of the concepts in its ecosystem, such as Git and GH. These concepts are also relevant to related services such as GitLab and BitBucket.

GHC provides a view of GH that is intended to be useful for instructors and students. GH is a Git repository hosting service. So the starting point to understanding GHC is Git.

*Authors listed in reverse alphabetical order.
[1]https://git-scm.com
[2]https://github.com
[3]https://classroom.github.com

ACE '22, February 14–18, 2022, Virtual Event, Australia

[4]https://gitlab.com
[5]https://bitbucket.org
[6]https://about.gitlab.com/solutions/education
[7]https://bitbucket.org/product/education
[8]https://education.github.com
[9]https://education.github.com/pack

2.1 GIT

Git is a Version Control System (VCS) [19]. It provides support for storing files and tracking any changes. The files, their histories, and any other information is grouped together in "repositories" (usually abbreviated "repo"). Unlike most earlier VCSs, which stored the repos in a central location, Git is a *distributed* VCS, a type of version control where the complete codebase—including its full version history—is mirrored on every developer's computer. A remote repo will be created for a project, and any developer in the project team will own their "local" copy by "cloning" the remote repo. As with any VCS, developers must notify Git of any changes they made to the files in the repo, a `commit`. This only updates the local repo. The developer must `push` the changes to notify the "remote" repo and must `pull` to learn of any changes made by other developers.

Because developers can change their "local" copy of the repo independently of each other, when independent streams of work need to be combined there is the possibility for the same file to be changed in inconsistent ways, resulting in "merge conflicts". Git (and most VCSs) provides mechanisms for understanding and resolving these conflicts.

Distributed VCSs bring several advantages over centralized VCSs (e.g., SVN[10]): (i) they are resilient to data losses as each developer has their own local copy of the repo; (ii) excluding `pull` and `push` operations that communicates on the remote server, all other operations on the repo are very fast because are done locally; (iii) they facilitate non-linear development.

One of the biggest advantages of Git is its *branching capabilities*. Unlike centralized VCSs, Git branches are fast to create and easy to merge. This facilitates non-linear development where developers creates branches to work concurrently and independently on certain features and eventually merge them in the main line of development (e.g., the `main` branch).

The popularity of Git has lead to many tools and techniques to improve its effectiveness and usefulness, such as integration with IDEs, and Desktop applications. It has also lead to the development of cloud services that provide (ironically) a central hosting service for repos, such as GitLab, BitBucket, and GH.

2.2 GitHub (GH)

GH is the most popular development platform with a community of more than 65 million users. The platform has more than 200 million repos and it is considered to be the "main hub" for Git Version Control.GH provides a web-based graphical visual interface that help developers to both manage their own repos and to browse other people's repos. GH is also a social platform where developers showcase their project and the community learns and grows together by reusing or learning from each other code [9].

Services such as GH, as well as providing hosting services for Git repos, provide additional services for managing those repos and for supporting collaboration. In particular, GH provides the **pull request (PR)** mechanism.

An issue with the `push` operation is that it updates the remote repo without any control from anyone on the project team. This can lead to unanticipated issues. To avoid this, instead of pushing changes to the remote repo, the developer can instead issue a PR. This effectively asks the maintainer (typically delegated to senior developers in the team) to do a `pull`. The maintainer can first review the proposed changes, that is, carry out a *code review*. The changes may then be approved (and so pulled into remote), or the developer may be asked to revise the proposal.

GH (along with other hosting services) provides support for *template repositories*. This allows for new repos to be created with useful content that is common to many new projects (e.g. organisational project management support).

Another important service of GH is GH Actions. This is a mechanism that allows scripting of actions to be taken after a specified Git or GH event has occurred. In particular, GH Actions can be used to set up a *continuous integration* (CI) workflow. For example, when pushing new commits to GH, a workflow can be set up to compile and then run tests over the new code.

GH provides several products. Importantly for teaching, it offers GH Free user accounts. These accounts are free, and offer quite generous support, including unlimited repos. This means that students face no barriers to getting a GH account. When students sign up for an account they choose the account name.

2.3 GitHub Classroom (GHC)

GHC is *"a teacher-facing tool that uses the GitHub API to enable the GitHub workflow for education"*[3]. GHC was created to facilitate and improve the use of Git and GH for education. GHC adds yet more features to GH. In particular it allows the creation of "classrooms" and "assignments" within classroom. Classrooms provide the means to organise assignments for a course, repos for the assignments, and students to the repos they use for the assignment.

The assignment management process is as follows. When the instructor creates an assignment, it must be associated with a visible repo (either it is public, or from the same GH organisation the classroom belongs to). The repo must be a *template* repo. The instructor names the assignment, and makes other decisions for the assignment, such as whether students have admin rights, or the repos are public (see Section 5.9).

Once the instructors create an assignment, GHC generates an *invitation link* (URL). This link is then made available to the students in the course. When they accept the invite, they get their own clone of the assignment repo. They can then complete their assignment with the contents of the (template) repo as the starting point, meaning that all repos have consistent names and consistent file structures. Crucially, the names of the repos are based on the assignment name and the students' GH account names, and instructors have *complete access* to all of the repos (See Section 5.4).

3 RELATED WORK

There has been an interest in computing education to use shared online storage and version control systems [3, 11, 17]. These online systems provide students tools for collaboration, version control, and storage [11, 24, 30].

One of the motivations for using tools such as GHC in computing education has been increased in team-based programming assignments [25, 27]. GHC provides students and educators with the ability to manage (and assess) multiple users contributing to and collaborating on the same project [2].

[10] https://subversion.apache.org

Table 1: Course details

Course*	Discipline	Level	Topic(s)	Year(s)	# Students	Git	Assumed Git XP†	Git-based Assessments‡
COMPSCI 331	Computer Science	3rd Year	Large-scale Software Development	2021	93	GHC	Basic	6
COMPSCI 399	Computer Science	3rd Year	Capstone Course	2021 S1+S2	36,120	GH	Expert	1
COMPSCI 701	Computer Science	Graduate	Software Maintainability	2020,2021	25,67	GHC	Expert	4
COMPSCI 718	Computer Science	Graduate	Programming for Industry	2021	37	GHC	None	4
COMPSCI 719	Computer Science	Graduate	Programming with Web Technologies	2021	65	GH, GHC	None	14
SOFTENG 206	Software Engineering	2nd year	Software Engineering Design	2019-2021	97,106,129	GH, GHC	None	4
SOFTENG 281	Software Engineering	2nd year	OOP & DSA	2021	270	GHC	None	5
SOFTENG 325	Software Engineering	3rd year	Software Architecture	2021	140	GHC	Expert	6
SOFTENG 701	Software Engineering	Graduate	Software Quality	2021	95	GH, GHC	Basic	5
SOFTENG 750	Software Engineering	Graduate	Software Tools & Techniques	2021	150	GH	Expert	1
SOFTENG 751	Software Engineering	Graduate	High Performance Computing	2019,2020	69-45	GH, GHC	Expert	1
SOFTENG 754	Software Engineering	Graduate	Software Requirements	2017-2021	65	GH	Expert	3
COMPSYS 202	Computer Systems	2nd year	OOP & DSA	2020	195	GHC	None	5
MECHENG 270	Mechatronics	2nd year	OOP & DSA	2020,2021	105, 114	GHC	None	5
INFOSYS 221	Information Systems	2nd year	Programming for Business	2021	40	GHC	None	9

* Courses have been anonymised
† Assumed Git experience: *None:* expect to teach students everything they will use, *Basic:* expect to teach some things, *Expert:* expect to teach nothing.
‡ Assessments include assignments, projects, and lab exercises.

There are several studies aiming at assessing whether the use of VCSs make teaching programming courses more effective.

Hsing and Gennarelli conducted a study on 7,530 students and 300 instructors to understand if GH provides better learning experience [13]. The authors compared the results of students and instructors who use GH in the classroom with those who did not. The authors found that students who used GH in the classroom also felt they were more likely to take a similar course in the future than students who did not use GH in the classroom.

Through using GHC in an Operating Systems class, Kertész reports that students preferred the collaborative platform [17]. Some of the reported benefits of using GHC includes: (i) learning from each others faults (ii) getting help from colleagues much faster, even if they are at distance (iii) learning a development platform commonly used in industry (iv) making them aware of different solutions to the same problem leading to better decisions on how to implement certain tasks. Kertész also reported that students reflected a higher learning curve, which was due to learning both version controlling (Git) and the GH platform while focusing on the assignments from the class.

Haaranen and Lehtinen found that 92% of their students (N=141) reported that using GHC was a valuable learning experience [12]. Haaranen and Lehtinen included tutorials in their lectures on how to use Git, which appears to have contributed to the learning experience. Glassey surveyed eight tools that aims to overcome the technical barrier of Git and GH within an educational context [10].

All of these studies provide evidence that using Git, GH, and GHC in programming assignments is useful. Not only from a pedagogical point of view (being Git and GH popular programming tools that every STEM student should learn), but also to foster a productive environment for both students and instructors. In contrast, we focus on issues that we encountered and discuss how we addressed them.

4 COURSE CONTEXTS

We have used Git and related systems in a variety of courses, for different subjects, different class sizes, different levels of experience, and different contexts. Each variation has its own challenges, and require different approaches. In order to understand the lessons we present in the next Section, it is useful to know the context that our lessons come from.

Table 1 overviews key "demographics" for the courses in which the authors have used either GH or GHC. This includes a variety of courses, including undergraduate to graduate level, varying class sizes, and assumed Git experience. Section 5 discusses the many lessons learned from these experiences.

5 LESSONS LEARNED

We present our collective experience with this variety as a set of lessons we have learned.

5.1 Students are Students

One of the major challenge in teaching is that students are students. Many students do not follow instructions [1, 21]. Variable prior experience between students is also a challenge [4, 6, 16, 29]. Students with little prior experience using a VCS, can be at a disadvantage, but students with experience may overestimate their abilities and make careless errors [7, 22].

While none of this will come as a surprise to many instructors, the complexity of Git seems to magnify the consequences. In particular, any exercise or assessment that required students to closely follow instructions to be successful often lead to problems when they did not, and the kinds of problems from incorrect use of Git can be time-intensive for instructors to resolve. Some of what we have learned is understanding what kinds of problem might occur and various mitigation strategies.

5.2 Provide Git Instruction to Novices

In several of the courses where Git has been adopted, students in those courses are novices—either for programming and software engineering in general, or with Git specifically. We found that students tended to become relatively comfortable with use of Git for individual work, where the workflow simply involves cloning/pulling from a remote repo, making edits, committing and pushing those changes. However, even with these simple workflows, we noticed that students would often commit files that are best left out of repos, such as generated files (e.g. compiled code) and IDE configuration files. Git provides a mechanism for avoiding this—`.gitignore`—but many students seemed unaware of this feature (or even aware it is needed), or not comfortable with updating it. In our experience, it is best to provide students with a `.gitignore`, and instruct them on its use and purpose.

> **Key Lesson 1:** We should provide novice students with a `.gitignore` file and explain its purpose so that they do not commit unnecessary files to their repos.

5.3 Choose an Appropriate Git Workflow

Novice Git users tended to experience significant difficulty when using Git for teamwork for the first time. We found that simply referring novice students to documentation on a particular Git workflow (such as feature branching [14] or fork and pull [15]) was insufficient. Student teams would often become stuck with many merge conflicts and `main` branches in inconsistent states, requiring the intervention of an instructor to fix. For novice students, teamwork went much more smoothly in the cases where we mandated the use of one particular Git workflow (as opposed to allowing students to choose their own), and provided a "mini-project" designed to give teams practice with the workflow prior to commencement of the main project. Even so, issues with Git (and particularly, in resolving merge conflicts) proved to be one of the most significant drains on tutor and instructor time in these courses.

With regards to the choice of workflow to use for novice teams, we recommend feature branching [14]. Other workflows such as fork and pull [15] promise additional layers of protection against accidentally corrupting the shared `main` branch. However, in our experience, students experience far more issues due to these workflows' higher complexity. The relative simplicity of feature branching outweighs any potential benefits of fork and pull for students at this level. However, care must be taken to guide students away from adopting poor practices such as long-lived feature branches.

> **Key Lesson 2:** The feature branching workflow is a good option for novice student teams as it is simple and easy to use.

Sometimes attempts to address one issue creates another. In one course, we created a lab to help students learn about merge conflicts and how to resolve them. It is non-trivial to deliberately create a merge conflict, and so the lab instructions required a series of somewhat artificial steps by different team members, in particular in when branches should be created and named. While these students had some basic Git experience, for the most part they had not worked in teams. What we only learned much later is that some teams concluded that the instructions on branch creation and naming is how branches should be used in general.

It is worth noting that the issues in the preceding paragraphs apply only to students with little prior Git experience. Students in our more advanced courses, having gained experience in earlier courses and industry internships, generally showed the ability to successfully manage complex collaborative team workflows.

> **Key Lesson 3:** Proper instruction needs to be provided on how to use Git, and its best practices. The instruction should be commensurate with students' prior experience.

5.4 Assessment and GHC

The use of Git opens possibilities for how the assessment process can be conducted. GH and GHC offer even more possibilities. In its simplest form, submission from the student side consists of keeping their repos up-to-date (not always easy — students will be students). This reduces many of the common problems made by students (e.g. submitting on an LMS to the wrong course, or submitting the wrong assignment or older versions of the assignment).

There are a number of options for instructors to assess the repos. Provided instructors have access (Section 5.16), it is possible to clone every student's repo. As well as the state of their project at the time the clone is made, instructors then have access to the entire commit history. This provides options for assessment beyond just based on the source code. A summary of what is available, including what has been used by the authors, is below.

GHC autograding. GHC provides the option, called "autograding", to apply automated tests to student submissions. This is based on GH Actions (see Section 2). Although many of our courses involved automated tests, none of the authors used the autograding feature. In part, this was because it seems quite limited in the kind of testing it supports and when the tests could be executed, but mostly because we already had our own automated tests from past delivery of the courses, and it was simpler and more powerful to use that.

From the students' point of view, GHC's autograder is somewhat closer to a "summative" assessment in that it only activates when pushing commits to the remote repo. Furthermore, for various undergraduate courses where students are new to Git, (successful) commits appear with pronounced red crosses denoting *failed test cases*, which may concern (novice) students thinking this denotes a *failed commit* (rather than failing test cases).

Offline automated processing. Many of the courses cloned the students' repos and then applied separately (and usually previously)-developed automated processes (typically automated tests but other processing may be done). This was particularly effective with GHC, as the file structure expected by the automated system could be specified in the template repo. Previously to using GHC, we had to rely on students following instructions to provide the correct file structure (See Section 5.1).

In some of the assignments, students are also provided with the automated tests included in the repos (possibly with only a partial set of the test cases provided to students). This not only guides students with "formative" assessment for their development, but

it increases the chances of students conforming to the expected automated testing setup. Instructors could also use additional test cases by overwriting the test file in the student repos.

Assessment via GH. All of the students' repos are available through the GH web interface. We found this provided a useful means to perform certain kinds of assessment (e.g. understanding the quality of commit comments). This was even more so with GHC, through its support of linking student repos (see Section 5.18).

***Use of GH* ACTIONS.** As noted above, GHC provides an autograding feature via GH ACTIONS. While no course used autograding, several used GH ACTIONS directly to support automated testing. This meant we could use existing test frameworks directly as a CI workflow.

For example, for some courses, the marking was done through Makefiles while others used Maven. Both cases could be handled by a simple (and almost identical) CI workflow. This had the particular advantage that the students could run the tests locally. GH provides feedback as to whether the CI workflow succeeded, which meant that students got confirmation that what they had submitted was what they intended.

Custom Tools. The structure used by Git is well understood, and there are a number of tools available to process this standard structure (for example GitPython[11]). This allows for the development of custom scripts or tools that can be applied to the student assignment repos, and produce spreadsheets with useful statistics (e.g., commit information such as times, sizes, and branch information). This data can be further visualised to get an understanding of students behaviour.

GH (and so GHC) also provides a REST API[12] that can be used to access not only the basic Git data, but also data specific to GH features such as pull requests. This provides another option for extracting data about students' behaviour that does not require a local clone of the students' repos. One use we made of this facility was to extract data that was available via GH, but was inconvenient to access. With our REST-based tool we were able to extract it and present it more usefully.

Allocating marks for Git usage. In some courses, students are allocated a modest amount of marks (e.g. 1-2% of the overall course grade) to encourage consistent Git usage. This provides many benefits, for both students and instructors. From an academic integrity perspective, this makes it more difficult for students to plagiarise as they are aware that committing final versions with poor evidence of "work in progress" is unacceptable – only drawing attention to their assignment. Many students have also benefited from their Git commits, in the cases of courses that required a separate upload of their code to a separate platform (e.g. LMS) but they had unknowingly submitted an incorrect version (but this and other issues are overcome when instructors take submissions directly from the repos).

[11] https://gitpython.readthedocs.io
[12] https://docs.github.com

> **Key Lesson 4:** There are many possibilities on how to assess students' assignments, each one has pros and cons. We found that custom offline automated systems are effective.

5.5 GitHub Actions Limitations

There are limits on GH ACTIONS usage with GH Free. The most important is that each execution of workflow consumes processor time. The limit for GHC is 2000 minutes of time per month per Organization. This seems very generous suggesting GH Free is sufficient for teaching. However, for one course (COMPSCI 331) we had integration tests that were computationally expensive. This still should not have been a problem as there were only about 15 teams and since they had the test frameworks available locally, the expectation was that the workflows would not be executed often.

Unfortunately students will be students, and at least one team, in the lead-up to the submission deadline, used the CI as their main test vehicle, exhausting the budget for the whole class in 2 days. While this did not prevent students from running the tests locally, or pushing their changes to GH, when they did so they received a message indicating the workflow had not executed. This was disconcerting, particularly in the hours immediately before the submission deadline. This was able to be resolved by purchasing (for a small amount) extra allocation. However there were upset students and an instructors' Sunday afternoon was interrupted.

In later courses we tried an alternative approach. We set up the workflow to only execute when the work was pushed to a specific branch, namely `submission`. We advised the students that they should do their development on `main` and then merge with `submission` when they were ready to submit. We reasoned the extra work involved (the merge) would discourage them from using `submission` until it was really needed. But students will be students and many just did all their development on `submission` (and see lesson 5.6 below).

We have since learned that the REST interface to GH appears to give us the ability to enable and disable workflows, and access the workflow usage data. With this, we may be able to develop a better way to manage the workflow budget.

An alternative is to use a product that provides a higher allocation, such as GH Campus Program[13], and this may be the best option for institutions that have committed to making significant use of GH. But for those who are still in the exploration stage, be aware of GH ACTIONS allocation.

> **Key Lesson 5:** There is a generous limit for use of GH ACTIONS, however it can still be used up in some circumstances.

5.6 Branches in Template Repositories

A GHC assignment must be created based on an existing *template* repo. As noted in Section 5.15, this has the benefit of allowing instructors to specify a specific organisation (e.g. directory structure) for students to start with.

However, some forms of organisation cannot be specified this way. For example, if the students are unfamiliar with working with branches, it might be tempting to set up the repo with the necessary

[13] https://education.github.com/schools

branches in place (e.g. to address the GH ACTIONS allocation issue discussed in Section 5.5). Unfortunately a property of template repositories is that the branches in any repo created from them will have *unrelated histories*. This makes ordinary merging of branches impossible. The proposed solutions involve quite sophisticated uses of Git and we concluded having the students create their own branches was in fact less error prone.

> **Key Lesson 6:** Branches in template repos have unexpected behaviour, so it is better to let students create their own branches when they are needed.

5.7 Changing the Template Repository

Students get their own "clone" of the template repository specified in the GHC assignment, however it is not a clone in the usual sense. If the instructor changes the template, students who have already accepted the invite cannot just issue a `pull` and expect to see the changes. There is not direct connection between their repo and the original template. The only recourse seems to be to distribute the changes separately and have the students incorporate them manually. Students who accept the invite after the changes are made do see the changes however.

> **Key Lesson 7:** The template repository in GHC should be checked thoroughly for correctness prior to sharing it with students. It is cumbersome to make changes afterwards.

5.8 Pull Request Feature

One key feature of GHC is the ability to provide feedback to students on their work via a pull request. When an assignment is created on GHC, a feedback pull request can be automatically created for each student repo. Instructors can leave general comments or line-by-line comments on student code. In our experience, we find this feature useful for providing specific feedback to individual students when they have questions or bugs in their code. The instructors are able to check student work more effectively directly on GH, which is especially for online courses.

Prior to use of GH, working with students' code meant arranging a meeting time, receiving the code via email in zip files, viewing via share screen, and similar somewhat inconvenient options. With GHC, the instructors are now able to access individual repos and provide prompt feedback via the pull request feature.

The pull request feature is helpful for answering assignment questions. However, providing detailed feedback with line-by-line comments for every assessment can be time consuming and resource intensive for large classes. For example, in COMPSCI 719, the marker would take 10 to 30 minutes to mark a 10% coding assignment with thorough feedback. The assignment is estimated to take a novice programmer 10 to 15 hours to complete.

> **Key Lesson 8:** A pull request workflow where instructors have to accept PRs is a great way to give feedback on student's code. However, is not practical for large classes, given the high workload that it puts on instructors.

5.9 Give GH Admin Rights to Students

For our beginner courses (year 1 & 2) we prefer to not give admin rights to students on their repos as they are new to GH and giving them access to sensitive and destructive actions like managing security or deleting a repo is not safe.

However, in our advanced courses, such as the capstone course (COMPSCI 399) and graduate level courses (SOFTENG 701), we do give teams admin rights on their repos. In these courses, students require greater control on their repos and need to make several independent management decisions. Although students have admin rights on their repos, in some courses, we enforce restrictions on admin permissions at the organization level. For example, in COMPSCI 399, students cannot delete a repo, transfer it to another account or change its visibility (which is set to private).

The repos need to be private to avoid plagiarism, since sometimes multiple teams may work on the same project during a semester. Once students finish the course, they sometimes request the instructor to make their repo public (Section 5.14). In other courses, for example, SOFTENG 701, each team works on a different project and can have public repos from the onset, since plagiarism is not an issue in such cases.

> **Key Lesson 9:** While its better to not give admin rights to novice-level students, we found that students at experienced-level courses might need admin rights to make independent management decisions.

5.10 Branch Protection

In courses where students work in teams and all team members have full admin rights on their repos (such as in COMPSCI 399), we encourage students to enforce protection rules on important branches of their repos, such as the `main` branch and/or the *development* branch. GH provides *branch protection* rules which prevent a branch from accidental deletion or loss of commit history on the branch due to a "force push". We have observed that the most common branch configuration option that students use is to "require pull request reviews before merging" on the `main` and/or the *development* branch. This option ensures that a pull request cannot be merged into the protected branch before it receives a specific number (which is configurable) of approving reviews.

> **Key Lesson 10:** When students work in teams, it is preferable to enforce branch protection rules to avoid improper workflows.

5.11 Legacy `master` branch

GH recently change the default branch from `master` to `main`.[14] We encountered problems where our repos had been set up before the renaming took place and so used `master`, but after renaming tools such as IDEs expected `main`. While this is less of a problem now, we have also encountered students with legacy settings in their tools, and so we still need to deal with this.

[14] https://github.blog/changelog/2020-10-01-the-default-branch-for-newly-created-repositories-is-now-main

5.12 GitHub Organizations

When it comes to managing the multiple repos for courses, GH's notion of an **Organization** is the natural choice. The bigger question tends to be whether the Organization is per *course*, or per *course offering*. The authors have tended to define a new Organization for each course offering, mostly because of the clarity and granularity it brings with having a separate account encapsulating the relevant offering's repos, Teams, and instructor/TA access. Collectively, the separate Organizations have made their management easier while more easily allowing replication of assignment repo names. This structure and separation also mirrors what is typically seen in LMSs, in particular Canvas which is used at University of Auckland.

For the management of the Organization, the instructors are added with the *Owner* role. If TAs are also expected to access student repos, they also need to be added with the Owner role. Students, on the other hand, have the standard *Member* role (or even *Outside Collaborator*) to ensure they do not have access to any private repos. In the case of group-based assignments, a **Team** is created within the Organization, with the corresponding group of students. To provide access to the private repos, this is defined by the repo's *Manage access* setting, by either specifying students individually or as a Team with (usually) just *write* access.

If this seems like a lot of administrative work for an instructor to manage, it is. This is where GHC alleviates many of the manual process surrounding access to individual and group-based repos, and of course forking repos to follow a consistent naming convention. A Classroom in GHC is associated with a GH Organization, and instructors have the option to either reuse the same GH Organization across multiple Classrooms (i.e. multiple course offerings), or to simply associate with a new Organization each time. Behind the scenes, GHC is creating the repos, creating Teams, and assigning access to repos for the respective GH Organization. More on group-based assignments via GHC follows in Section 5.13.

> **Key Lesson 11:** GH's notion of an Organization better lends itself to a single *course offering*, as opposed to being reused over multiple offerings of the same course. This provides easier management and clarity, particularly for archival purposes.

5.13 Group Assignments with GHC

To facilitate the management of group-based repos, GHC provides the option (when creating the assignment) to specify whether it will be a *group assignment* or an *individual assignment*. When the *group assignment* option is selected, this in turn provides additional fields for the instructor to set:

"Name your set of teams" (required) is used for representing the set of teams. If a course has multiple group assignments, and each group assignment involves different groups, this field allows distinguishing the different sets of teams. When creating a group assignment and there is already an existing set of teams, then an option appears to select it rather than defining a new set of teams. If an existing set of teams is specified, this means students do not need to select their team when accepting the new assignment (as they will be in the same Team).

"Maximum members per team" (optional) is used to prohibit further students joining a particular team, given that team has reached the limit. If some teams have fewer members than others, this means others could potentially "peek" by momentarily joining them and then leaving from the GH Teams interface.

"Maximum teams" (optional) is used for prohibiting further teams from being created, when this limit is reached.

An *assignment title* is required for group assignments (as in the case of individual assignments); this is used as the prefix for the name of the generated repos. Also similar to individual assignments, no repos are forked at this stage. When a student accepts the invitation link to a GHC group assignment, the repo is forked, and they are presented with one of two situations. If the "*Name your set of teams*" setting referred to an existing set of teams (i.e. from a previous group assignment where students were already allocated), then students immediately proceed to accessing their new repo (in this case, another repo is created in the same GH Organization Team—see Section 5.12). In the situation that the "*Name your set of teams*" setting refers to a new set of teams, students will be presented with one of two options:

(1) If other students (not necessarily from the same team) have already accepted the invitation link before them, a list of existing teams will appear. The name(s) of the team(s) that appear in this list are defined in case #2 below. The student has the option to select *any* one of these existing teams (red-flag #1), regardless of whether it is their actual team.
(2) If the student does not see their team name in the list of existing teams of case #1 above (either because there are no existing teams, or they are the first member of their team to accept the invitation link), they can create a new team by specifying *any* name they wish (red-flag #2).

In the case of #2, a new repo is forked (and given the name "*<assignment-name>-<team-name>*"), along with a Team (given the name "*<team-name>*")—both of which are added to the respective GH Organization. The student accepting the link is added into that team, and can therefore access the new Team's repo.

It is important to recognise that GHC does not have any notion of which team students *should* belong to. As such, it relies on the students to (i) correctly initially define team names (so as the instructor can easily find and reference the teams' repo), and (ii) correctly select their team (so as not to gain access to another team's repo—until an instructor intervenes by removing them from the respective GH Organisation Team).

In the case of following a systematic naming convention, this is essential in simplifying the instructor's management of teams. To address this, the authors have resorted to one of two approaches:

(1) If there are few teams, the instructor can iteratively (i) accept the assignment link themselves, (ii) create a new team using the naming convention they desire (this now creates the Team and their repo), and (iii) remove themselves from that Team via the GH Organization interface. This needs to be repeated until all the teams are created. When a maximum number of allowed teams is specified in the GHC assignment settings, students will therefore not be able to define new teams, and are instead required to join one of the existing

teams. Although this approach results in the satisfaction of perfectly consistent Team (and repo) names, it is clearly tedious and unscaleable for a large number of groups.

(2) The alternative is to let students define their own team name, but provide guidelines on how to name their team. Regardless of how clear instructions are, students will be students—some groups are still likely to get this naming incorrect. What is worse, upon realising on their own (or pointed out by their peers) that they did not follow the naming convention, students will leave the Team (via the GH web interface), reaccept the assignment invitation link, and create another team with the correct name. The problem here is that the (now deserted) Team with the incorrect name is still claiming towards the quota of the "*Maximum teams*" field of the GHC assignment setting. The instructor will therefore need to go to the GH Organization, and delete the deserted Team to allow other groups to form their team via GHC.

Given that the instructor has already specified all the group assignment parameters in GHC, it would have been appreciated to at least offer the option to automatically pre-generate the Teams, using the name of teams as a prefix, followed by an incremented integer as the suffix.

Instructors need to also be careful of students joining incorrect teams, whether it was intended or otherwise. If no limit was specified for the (optional) "maximum members per team", or if a particular team had not reached its limit, students are able to join (and gain access) to another team's repo. What is more concerning, is that a student could easily accept the invitation link for one team, clone the team's repo, remove themselves from the GH Team, then repetitively accept the invitation again and again for different teams (that have not filled up) and effectively have a snapshot of many groups' repos.

Hopefully a future feature of GHC will be to allow instructors to predefine team allocations, by way of uploading CSV or connecting to the LMS's group information.

> **Key Lesson 12:** Setting up group-based assignments in GHC is not trivial. Many things can go wrong. Following a systematic naming convention and being mindful students may move between open GHC teams is essential in managing groups.

5.14 Repos Can Move: A Double-Edged Sword

With Git, it is possible to move[15] an entire repo, from one remote repo to another remote repo, along with the full history of commits (regardless of the Git cloud service(s) involved). From the context of a course, this has both useful and highly undesirable consequences. The ability to move repos is useful when students reach out to instructors at the end of the course, wishing to make their repo public or share with prospective employers as part of their resume portfolio. While this is easy to process through the GH interface, it is simply easier to direct students to the instructions of creating their own repo and moving (a copy of) their assignment to it. But here comes the undesirable consequences: it is possible for students to operate on their own (self-created and self-managed repo) for the assignment, and merely move it to the instructor-allocated remote repo at a later time, where it was unknown who else had access to their personal repo. Even with a non-empty instructor-allocated repo, students can still `-force push` to it. The only giveaway to detect this is when inspecting the commit logs, and recognising the original commit (e.g. the one from the instructor, or automatically generated by GHC) no longer exists.

> **Key Lesson 13:** A repo can be moved to another empty repo. This is great when the course ends and students want to grow their portfolio, but means students can force push other repos to it, raising potential academic misconduct concerns. Check that the instructor's initial commit is still there to detect such cases.

5.15 Choosing GHC over GH

GHC is intended to support teaching, and so there are benefits for using it compared to the basic GH service (but see Section 5.16).

For programming assignments there are several benefits of following the GHC assignment management process over simply use GH. Providing a template repo is useful because all the submissions will have a shared setting prepared by the instructors based on the characteristic of the assignment. This is particularly important if automated tests are used as part of the assessment process (Section 5.4). Such tests typically rely on all repos having the same structure. By starting with a common template with the expected structure, students submissions are more likely to have the correct structure.

Use of plain GH for assignments is possible, and there are good reasons for doing so (See Section 5.16), but it requires that students create their own repos in their GH accounts. When they create these repos they can choose arbitrary names, making them hard to identify, and structure it anyway they like. If they make their repos public, there is the possibility of plagiarism. If they make them private, instructors do not have access unless the students remember to invite the instructors to be collaborators in order for submissions to be made (or an alternate submission is needed). And then the instructors have to remember to accept the invitation in time (they expire after seven days). While clear and detailed instructions can be given, students will be students (Section 5.1). Our experience is that it is not uncommon for there to be problems with 5–10% of the submissions, which in courses with 200 or more students represents a significant cost to instructors.

In contrast, with GHC, the repos are created with a consistent naming scheme, based on a common structure, can be required to be private, and instructors are automatically made collaborators. The consistent naming scheme makes it easier to find a student's repo, and in fact GHC supports more usefully identifying the repo (See Section 5.18).

> **Key Lesson 14:** For undergraduate courses GHC would be a better choice than GH, as GHC simplifies the educational use of GH.

5.16 Choosing GH over GHC

When deciding to use GH for assignments, one decision is whether to use it directly or through GHC. The reasons for choosing GHC have been discussed (Section 5.15). There are also good reasons for choosing GH.

[15] For example: https://atlassian.com/git/tutorials/git-move-repository

One such reason is when we wanted students to learn and experience the use of **Continuous Integration (CI)** [20]. In our courses we experienced CI with both GH ACTIONS and hosted continuous integration services, such as Travis-CI[16]. Travis-CI is used to build and test software projects hosted on GH. Similarly to GH ACTIONS, Travis-CI seamlessly integrates with GH, automatically triggering a build and test of the code for every `push` on a watched branch (often the main branch). Travis-CI is currently the dominant cloud-based continuous integration service on GitHub, but there are not much differences with the younger and rising GH ACTIONS service.

The issues with GHC and CI is the available workflow budget (see Section 5.5). For instance, Travis-CI gives, for free, only one single-threaded build service for *all* the students. This is a problem when the deadline is approaching and all students are pushing at the same time. Also, from a pedagogical point of view students should learn how to setup, use, and leverage CI services during development. This will not be possible if GHC setups and manages all the Travis-CI builds on the students behalf. For these reasons, in the course SOFTENG 754 we had to rely on student-created private repos, because CI was a fundamental aspect of the course. By doing so, we experienced first hand all the disadvantages of student-created private repos discussed in Section 2.2. Hopefully, in the future will be possible to associate each GHC student repo to a single Travis-CI build. Notably, we had to ask students to request (for free) the GH Student Pack to use Travis-CI with private repos.

One other instance where we need to rely on student-created repos is in one of our capstone course COMPSCI 399. In this course, students work on different projects in teams of 7-8 people. There is no starter boilerplate code to share in such a scenario and the template repository-approach offered by GHC is not needed. Furthermore, the teams may create multiple repos based on their project requirements. For example, many projects are web-based applications and students prefer to create separate repos for the front-end and back-end code in such cases. Given the big team size, most teams divide themselves into smaller sub-teams that work separately on different parts of the code. The organization of code into separate repos on GH allows the sub-teams to define their own workflows.

In general, we found that in graduate level courses GH would be more appropriate over GHC. For example, in one of our graduate level courses (SOFTENG 701), each team works on a different project. Thus, academic integrity issues are no longer a concern. Students are encouraged to make their repos public, and this also helps students to build up their GH profile, which can be useful in their job applications. The use of GH over GHC also allows students to learn how to setup a repo from scratch, including creating appropriate documentation and setting up a `.gitignore` file. Another benefit is that students have control over all aspects of the repo, allowing them to make management decisions. This enables greater critical thinking for these more advanced students, who can consider the tradeoffs of different repo settings. Some of the problems discussed above, are also mitigated since each group has their own repo and the CI limits are not spread across the entire class.

[16]https://travis-ci.org for public repos, https://travis-ci.com for private repos

> **Key Lesson 15:** In graduate courses or higher stage undergraduate courses where students should have access to advance tools (e.g., CI) and management decisions, GH would be a better choice than GHC.

5.17 Submission Due Date

GHC enables the instructors to optionally set a deadline for each assignment. The instructors are able to access each student repo that contains the latest commit before the deadline via a separate "submitted" link. This feature is useful as the instructors no longer need to manually download each assignment repo on the due date. However, we learned that there is an ongoing bug with this feature. "not submitted" would show up for some repos even if the students have made some commits before the specified due date. We also learned that students are able to commit their code after the due date. As the bug is not yet resolved, we have to explicitly clone the repos on the due date. See Section 5.4 on our assessment process.

> **Key Lesson 16:** The GHC feature to set a deadline for assignments is not reliable (as of today), manually downloading the GHC repos on the due date is a better option.

5.18 Synching/Linking Accounts with LMS

While repos created by accepting a GHC assignment invite have a consistent naming scheme, part of the name is the *GH account name* chosen by the student (See Section 2.2). While the problem of finding a particular student's repo is made easier due to the consistent scheme, in a large class it can still take some effort. GHC addresses this problem by providing a roster for each classroom. Instructors can upload the list of the students in their course to the relevant classroom, and then the students' repos can be linked to the roster. This allows repos to be found using the student's University identity.

There are two ways for the instructors to create the roster of the students in their courses. The instructors can manually import a roster by uploading a CSV or a text file that contains student identifiers as student ID. Alternatively, the instructors can import the roster from a Learning Management System (LMS) such as Canvas or Moodle. To associate the students with their GH accounts, students must identify themselves from the roster and manually link their accounts when they first accept the GHC assignment.

From our experience, we find that syncing the roster directly from Canvas is fast and convenient. However, there are several caveats. The first caveat is that the connection between the LMS and GHC does not automatically link students' GH accounts with the GHC roster. Students are still required to manually link their accounts. If students accidentally or maliciously pick the wrong name from the roster, then the instructors would need to manually unlink the accounts.

Another caveat is that there are only three options for identifying each student in the roster: User IDs, Names, Emails. We use emails as the student identifiers as some students might have the same names in the same course and the user IDs imported from the LMS are not always equivalent to the student IDs. Sometimes, we find that the students are unable to find their emails from the roster if the class is large. Also, it is not just students that appear in the roster.

Anyone who appears on the LMS class list, which may include instructors and other support staff, appears in the GHC roster. This can create confusion, for example when trying to determine which students have not yet engaged with GHC.

To manage the roster in GHC, especially for large classes, we think manually creating and uploading the list of the students would be a better approach. To minimise confusion for the students and the instructors, we use a combination of names, usernames, and student IDs as student identifiers. Notably, there is no need to create the list of students from scratch: instructors can extract a roster from LMSs in CSV and revise it accordingly. The tradeoff is that the roster must be adjusted should the course enrolment change.

> **Key Lesson 17:** We found that using the GHC feature that connects to LMSs to import student roster has some caveats. It is often better to manually create and import CSV files for the students' rosters.

6 CONCLUSIONS

The decision of whether to introduce a new technology into a course is difficult to make at the best of times, but it is especially difficulty when the technology is as complex as Git, where when things go wrong they can go really wrong. While there is an enormous amount of documentation on Git, and related systems such as GH and GHC, it is so enormous that finding the information needed for the decision is hard. And often the information does not contain the "gotchas" that only become evident through actual use. This paper is meant to help fill that gap.

In this paper we have presented the hard-won lessons regarding using GH in the Classroom based on the experience of 8 instructors successfully teaching 15 distinct courses, 6 with multiple offerings (25 offerings in total). What we can say is, there will be additional teaching load when introducing GH to the Classroom, as is the case when any new technology is introduced, but our conclusion is that it is worth it. Once the issues are worked out, GH can add significant value to courses. It helps instructors with development of assignments and management of student submissions. It helps students with working on the assignments and submitting them. They also really appreciate the use industry-standard tools in an academic context. We hope the lessons we have presented help others with making the decision to, and the use of, GH in the Classroom.

REFERENCES

[1] Raad A Alturki et al. 2016. Measuring and improving student performance in an introductory programming course. *Informatics in Education-An International Journal* 15, 2 (2016), 183–204.

[2] Berk Anbaroğlu. 2021. A collaborative GIS programming course using GitHub Classroom. *Transactions in GIS* (2021).

[3] Miguel A Angulo and Ozgur Aktunc. 2019. Using GitHub as a teaching tool for programming courses. In *2018 Gulf Southwest Section Conference*.

[4] Susan Bergin and Ronan Reilly. 2005. Programming: factors that influence success. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*. 411–415.

[5] Esmail Bonakdarian. 2017. Pushing Git & GitHub in undergraduate computer science classes. *Journal of Computing Sciences in Colleges* 32, 3 (2017), 119–125.

[6] Pat Byrne and Gerry Lyons. 2001. The effect of student attributes on success in programming. In *Proceedings of the 6th annual conference on Innovation and technology in computer science education*. 49–52.

[7] Dennis E Clayson. 2005. Performance overconfidence: metacognitive effects or misplaced student expectations? *J. of Marketing Education* 27, 2 (2005), 122–129.

[8] David C Conner, Matthew McCarthy, and Lynn Lambert. 2019. Integrating Git into CS1/2. *Journal of Computing Sciences in Colleges* 35, 3 (2019), 112–121.

[9] Mohammad Gharehyazie, Baishakhi Ray, and Vladimir Filkov. 2017. Some from here, some from there: Cross-project code reuse in GitHub. In *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 291–301.

[10] Richard Glassey. 2019. Adopting git/GitHub within teaching: A survey of tool support. In *Proc. of the ACM Conference on Global Computing Education*. 143–149.

[11] Ángel Manuel Guerrero-Higueras, Vicente Matellán-Olivera, Gonzalo Esteban Costales, Camino Fernández-Llamas, FJ Rodriguez-Sedano, and MÂ Conde. 2018. Model for evaluating student performance through their interaction with version control systems. *Proc. of the Learning Analytics Summer Institute Spain* (2018).

[12] Lassi Haaranen and Teemu Lehtinen. 2015. Teaching Git on the side: version control system as a course platform. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. 87–92.

[13] Courtney Hsing and Vanessa Gennarelli. 2019. Using GitHub in the classroom predicts student learning outcomes and classroom experiences: Findings from a survey of students and teachers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 672–678.

[14] Eirini Kalliamvakou, Daniela Damian, Kelly Blincoe, Leif Singer, and Daniel M German. 2015. Open source-style collaborative development practices in commercial projects using GitHub. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 574–585.

[15] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining GitHub. In *Proc. 11th working conference on mining software repositories*. 92–101.

[16] Zoe A Kersteen, Marcia C Linn, Michael Clancy, and Curtis Hardyck. 1988. Previous experience and the learning of computer programming: The computer helps those who help themselves. *Journal of Educational Computing Research* 4, 3 (1988), 321–333.

[17] C. Kertész. 2015. Using GitHub in the classroom - a collaborative learning experience. In *2015 IEEE 21st International Symposium for Design and Technology in Electronic Packaging (SIITME)*. 381–386. https://doi.org/10.1109/SIITME.2015.7342358

[18] Joseph Lawrance, Seikyung Jung, and Charles Wiseman. 2013. Git on the cloud in the classroom. In *Proceeding of the 44th ACM technical symposium on computer science education*. 639–644.

[19] Jon Loeliger and Matthew McCullough. 2012. *Version Control with Git: Powerful tools and techniques for collaborative software development.* " O'Reilly Media, Inc.".

[20] Mathias Meyer. 2014. Continuous integration and its tools. *IEEE software* 31, 3 (2014), 14–16.

[21] Daniel M Oppenheimer, Tom Meyvis, and Nicolas Davidenko. 2009. Instructional manipulation checks: Detecting satisficing to increase statistical power. *Journal of experimental social psychology* 45, 4 (2009), 867–872.

[22] Vennila Ramalingam, Deborah LaBelle, and Susan Wiedenbeck. 2004. Self-efficacy and mental models in learning to program. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*. 171–175.

[23] Arnold Rosenbloom, Sadia Sharmin, and Andrew Wang. 2017. Git: Pedagogy, use and administration in undergraduate CS. In *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*.

[24] Matthew NO Sadiku, Sarhan M Musa, and Omonowo D Momoh. 2014. Cloud computing: opportunities and challenges. *IEEE potentials* 33, 1 (2014), 34–36.

[25] Pilar Sancho-Thomas, Rubén Fuentes-Fernández, and Baltasar Fernández-Manjón. 2009. Learning teamwork skills in university programming courses. *Computers & Education* 53, 2 (2009), 517–531.

[26] Diomidis Spinellis. 2005. Version control systems. *IEEE Software* 22, 5 (2005), 108–109.

[27] Diomidis Spinellis. 2012. Git. *IEEE software* 29, 3 (2012), 100–101.

[28] Stack Overflow. 2021. 2021 Developer Survey. https://insights.stackoverflow.com/survey/2021.

[29] Harriet G Taylor and Luegina C Mounfield. 1994. Exploration of the relationship between prior computing experience and gender on success in college computer science. *Journal of educational computing research* 11, 4 (1994), 291–306.

[30] Ewan Tempero and Yu-Cheng Tu. 2021. Assessing Understanding of Maintainability using Code Review. In *Australasian Computing Education Conference*. 40–47.