2021

Improving Grading and Feedback of Programming Assignments Using Version Control: An Experience Report

Jillian Morgan
Georgia State University

Michael Weeks
Georgia State University

Improving Grading and Feedback of Programming Assignments Using Version Control: An Experience Report

Jillian Morgan and Michael Weeks

***Abstract*—Leaving meaningful, actionable feedback that students will read and, most importantly, follow-up on, is essential for strengthening their programming skills. In addition, being capable with version control platforms, such as `git`, is a desired skill in industry. Could a marriage between the two, leaving meaningful feedback for student submissions in a version control system, lead them to be better programmers while improving the time and quality of instructors' feedback? This experience report describes how we used GitHub Classroom for programming assignment submission and assessment in CS2. We provide examples of typical feedback using various assessment mechanisms, describe the process of assignment submission for students, the assessment process for instructors, and reflect on students' reception towards the process and the value, in terms of time and quality, for the instructor.**

I. Introduction

Programming assignments are the necessary evil of computer science in academia. After all, the best way to learn programming is by doing. Programming assignments provide a way for students to practice and hone their skills. However, they are very time consuming to grade, especially when the desire is to provide feedback that gives students insight and considerations for improving their skills. According to [1], the components of grading a programming assignment are whether the

1) program produces the desired output
2) tasks are performed without using too much processing time and memory space
3) code is easy to understand and, as such, maintain.

When examining programming assignments for these components, providing useful, easily understandable feedback is paramount as it helps students identify weaknesses in their skills.

In addition to growing their programming skills, computer science students could also greatly benefit from the addition of industry-standard tools and technology to our curriculum. One such tool is a version control system called `git`. Introducing students to a professional tool will allow: students to work on multiple machines (university-owned or personal), instructors and TAs to have an up-to-date copy of students' work for more meaningful office hours, and patterns in student programming to be identified [2]. It would also give students an edge when interviewing for competitive positions and a solid foundation once hired. Even though using version control systems is an important skill, it is usually not included in a CS curriculum [3].

In this paper, we propose a method for improving the time and quality of feedback on programming assignments using a popular `git`-based distributed version control system, GitHub [4]. It provides a complete version history for tracking changes to documents.

The rest of this paper is organized as follows. An introduction to grading programming assignments and version control systems is in Section II. Section III explains the assignment setup and submission and feedback processes. Our experiences introducing `git` for programming assignment submission and feedback are discussed in Section IV, leading to our conclusions and future work in Section V.

II. Background

In order to understand the impact of using `git`-based version control for grading programming assignments, it is first helpful to understand how programming assignments are typically graded and how version control systems work.

A. Grading Programming Assignments

As mentioned earlier, programming assignments are notoriously tricky and time-consuming to grade while providing one of the best ways for students to practice their programming skills. For this reason, many instructors, especially those with large class sizes, are moving towards automated or, at least, semi-automated grading tools.

Cheang et al. created an automated grading tool called Online Judge, modeled on the way programs are tested during the ACM International Collegiate Programming Contest. It uses "simple string matching" [1] to determine whether the student's output matches predetermined output provided to the tool and used memory space and processing time efficiently. The focus of [5] is on providing quick, 24-hour, objective assessment for students. Students can use the tool at any time while working on their assignments to receive written and numeric feedback on their assignments to ensure they are on the right track. Adams discusses his tool, Aristotle, in [6]. Aristotle is an open-source, semi-automated program grading tool for senior-level undergraduate and graduate courses that allows an instructor to import and build and test submissions and generate a report that can give the instructor more context to evaluate the code manually. It is worth noting that Aristotle can be integrated with GitHub's education tool for instructors, GitHub Classroom [7]. This tool allows institutions to create

assignments that students can submit to GitHub. For a detailed study on automated grading tools, see [8].

Automatic grading is not a fail-safe system, however. As Adams mentions, it is not possible for a computer to assess some of the criteria for an assignment in a meaningful way [6]. Automated grading can only focus on whether the submitted program produces predetermined output and whether the tasks are performed efficiently in some cases [1]. The automated system cannot determine whether the method used to get to this output was correct for the assignment given. For assessing such tasks and providing meaningful feedback for a student to improve his programming skills, some instructor intervention is required.

This intervention can take the form of merely writing descriptive comments for the assignment in a Learning Management System (LMS). This does not always provide feedback that is easy to follow, as it is somewhat detached from the lines of code. Feedback for programming assignments should be tied to the line itself so that students can find their exact problems to fix to improve their programming skills.

Intervention can also be done as in [9] where Yan et al. describe their programming assignment submission tool, Pensieve. It allows teaching assistants (TAs) and instructors to provide formative feedback to be given on assignments in progress rather than summatively at the end of the assignment. It is used to facilitate a conversation about the process by "thoughtfully" integrating human feedback which is essential for introductory programming students. The primary focus of Pensieve is on large CS1 courses with in-class and grading support, with less focus on how to use VCSs [9].

One instructor intervened technique that we employ to provide this feedback on specific lines of code is to convert programming assignment submissions into a PDF that can then be annotated. Fig. 1 shows an example.

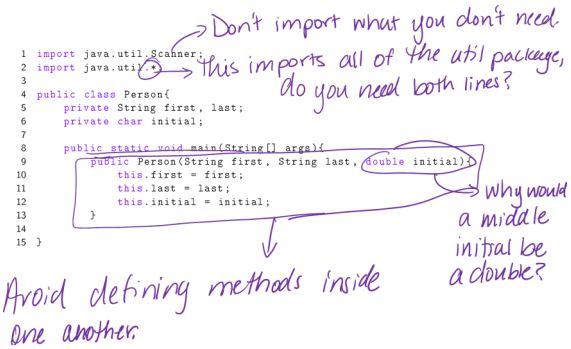
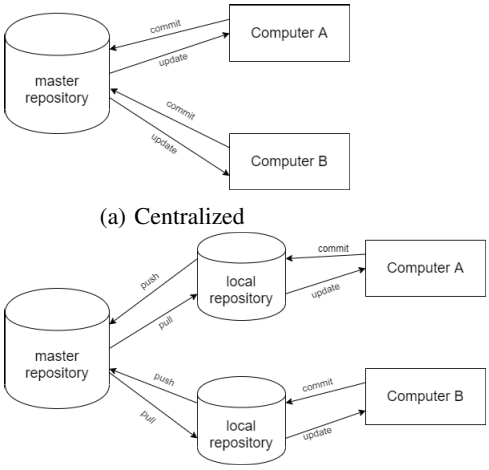


Fig. 1: Example Annotated Feedback

Even for a small and short example like this one, providing this type of feedback is still very time consuming and does not provide the opportunity for students to gain experience using industry-standard technology such as version control systems.

B. Version Control Systems

Anyone who creates documents has used rudimentary version control at one time or another. This typically takes the form of saving documents with naming conventions such as `proposal.doc`, `proposal_final.doc`, or `proposal_050120.doc`. In the computing industry, programmers use version control systems (VCS) to aid this by not only keeping track of the different versions of a file, but also of what was changed on each iteration. Version control systems can either be centralized or distributed. A centralized version control system stores the repository on a remote server (known as the *main*). Meaning that all changes are made directly to the master; this is known as a *commit*. A distributed version control system, on the other hand, employs the same fundamental model with one key exception; rather than a single copy of a remote master repository that each developer is sharing, each developer has their own local version of the master copy, separate from the others. When a developer wishes to work on the code in the repository, he downloads (called a *pull*) the code to his local machine. When ready, he makes the changes locally, then uploads the changes back to the remote server (called a *push*). If he is ready to change the master repository, he can issue a request to merge the changes from his remote repository to the master. A comparison of these systems is illustrated in Fig. 2.



(a) Centralized

(b) Distributed

Fig. 2: Centralized vs. Distributed Version Control System

In a collaborative environment, like the programming industry, a VCS also keeps track of which team member made which changes. In recent years, there has been a growing desire in the industry for students to be able to use VCSs when they enter the workforce. In April 2018, a recent graduate talked to current students and asked what she wished she learned, she said "`git`", without hesitation. Additionally, several leaders from a company with which we are partnered have expressed a wish for interns to experience `git`-based tools. GitHub and many other VCSs use the `git` system as their backbone. `git` provides commands for working with the VCS that students and instructors need to learn to use the system.

We know that this is useful for students, but what benefit do instructors gain from going through the process of, in most cases, learning `git` themselves [10], and teaching these commands to students? We argue that instructors would see an improvement in their workflow when it comes to grading programming assignments.

Studies on using `git` in the classroom have been ongoing for the past several years. Based on anecdotal reports from students interviewing for or being placed in internships, Kelleher [11] explored teaching `git` to second- and third-year students, before their participation in an internship. The students, with no prior knowledge of `git` or VCSs, worked in groups of four on a software project from inception to deployment and utilized GitHub to manage the project. The focus for the second-year students was to learn to clone repositories. They used a GUI application to clone roughly 80% of the exercises needed to be completed. The third-year students, on the other hand, used a command prompt and their goal, in addition to cloning repositories, was to create their own repositories, commit and push to remote repositories, and create and checkout branches (a divergence from the main line of commits to work on without messing with the main line of commits). Kelleher [11] discovered numerous benefits from utilizing `git` and GitHub in this way:

- secure assignment submission much like an LMS
- possibility to see student work as it progresses
- students were able to better organize their source files by using a VCS
- use of a `gitignore` file allowed less cumbersome file submissions
- the tool encouraged students to manage their code more diligently
- several students used `git` for projects in other courses as well, driven by the fact that they were using an industry-standard tool

Haaranen and Lehtinen [3] discuss how version control could be taught in a web development course, evaluated from both students' and instructors' point of view. Their study uses a course with 225 third-year students. Recommended prerequisites for this course were CS2 and a database course. Though this was merely a recommendation, it would mean that some, if not most, of the students had some coding experience and could be taught more advanced `git` features and commands. A webservice running GitLab (a service similar to GitHub) and their LMS was used rather than hosting assignments on GitHub itself. Data was gathered from a feedback survey administered to students and usage data from their forked repositories. 92% of the respondents said they had a positive attitude towards using `git` in the course. 72% of the students responding to the survey did have at least some `git` experience prior to starting the course. Haaranen and Lehtinen also noted that the instructors found their workflow simplified by using `git` to manage course assets.

Gowtham discussed implementing `git` as part of graduate and undergraduate computational sciences and engineering curriculum in 4000-level courses. Gowtham noted that the students were better prepared for real-world expectations and the instructor was better able to give timely feedback to students while also being able to monitor their progress on projects and assignments. Gowthan noted that "adopting `git` gave the teacher and opportunity to distribute the course materials to students with relative ease and provide timely feedback to partial submissions." The majority of students understood version control and the need for it, but did not want to learn newer technology [12].

Clifton et al. investigated adding version control to an introductory CS course (CS1) to "enhance course management." The focus was on Subversion, a centralized VCS, and on how the instructors and teaching assistants from two sections of a CS1 course could improve their workflow and "reduce administrative demands." They found that using version control allowed "instructors to provide direct and timely feedback to students." [13]

In an earlier study [14], Zagalsky et al. gathered data from the instructors' experiences using `git` in the classroom. The instructors who were interviewed used GitHub as a way to host course-related content and for students to submit assignments. In hosting course content, they found that students were able to gain a sense of community as they could suggest revisions to course content via *pull requests*, a way to ask collaborators to review your changes. They also found that there was transparency in how active the students were via several different features in GitHub, which encouraged participation. Most importantly, the instructors noted that GitHub has some of the more essential features that an LMS has, like assignment delivery and submission, but lacks other essential features like grading management tools.

Rosenbloom et al. [15] discuss the use and administration of `git` in CS undergraduate courses using an in-house installation of `git` rather than an already established service like GitHub. They had students submit and organize group work. The students would tag a commit as the one they wanted to submit, then the instructors/graders would clone and check out that commit and create a new feedback branch for disseminating feedback.

Most of these studies [12], [3], [11] were done with upper-level courses where students have a more mature understanding of programming languages and are more likely to grasp the commands faster and feel more comfortable with them. The studies that did use `git` or GitHub for assignment submission either only provided feedback on partial submissions [12] or did not mention how doing so affected grading workflow [14], [15].

Reid and Wilson [2] discuss moving a class from a command-line or web assignment submission process to Concurrent Version Systems, an early VCS that stores the history of a single file rather than a whole project like `git`-based VCS does. After their study, they were convinced that it should be introduced during a students second year to get working knowledge of industry standard tools early on, but do not state whether there is a benefit in the workflow for instructors. We intend to show that grading with `git` improves the workflow for instructors regarding assignment submission and grading.

III. Using Version Control for Improved Grading

This experience is still ongoing as we work to improve and streamline our workflow. The following discusses the basics of our workflow.

A. Assignment Setup

GitHub has tools and tutorials specifically designed for students and educators to learn new tools and technology via their GitHub for Education program [16]. One such tool is GitHub Classroom [7], which provides a way for students to use the GitHub VCS to submit assignments. Setting up an assignment using GitHub Classroom only takes a few minutes of clicking through their wizard.

The course section used for this study is at a small institution with small class sizes (between 30 and 40 students per section) and no TAs. We had about 35 students in one CS2 course section, comprised of both CS and IT majors and taught by a single instructor, use GitHub Classroom to submit programming assignments via `git`. It is worth mentioning that this institution requires students to have access to a laptop for class use and completing assignments.

There were three assignments total, each designed to take roughly 2-3 hours to complete over a span of 2 weeks, typical of assignments in our programming courses. Given the nature of the material, the first assignment was independent of the others. For this assignment they were given some starter code to give them structure and guidance, as well as some additional pointers on completing the assignment. The third assignment built off the second and the students were to review feedback on the second assignment and update their code before continuing on to the third. All assignments were to be done individually and were of moderate difficulty, though the students' perception is likely that the difficulty level was much higher.

Once the assignment was created in GitHub Classroom, a link was provided for students, creating a private repository (accessible only by the instructors and students) with the starter code included, or an empty one if no starter code is provided. Assignment instructions were given in the LMS like all other course content. The GitHub Classroom link was embedded in these instructions.

Near the beginning of the semester, the instructor spent about half of one class period (about 30-45 minutes) showing the students how to use the basic `git` commands using a terminal (command-line) window in order to understand the process and importance of version control. This lecture was recorded and a written summary of instructions was provided in the LMS for students to review as often as needed.

B. Assignment Submission

Students logged in to the LMS to get the instructions and the GitHub Classroom link to create the repository for the assignment. There were separate links for the first and second/third assignments, providing separate repositories.

They worked on the assignment on their own computers either in a text editor or one of our approved lightweight IDEs (i.e., JGrasp or Dr. Java). Once they were ready to commit to the repository, either periodically as they worked on the assignment or, most likely, at the end when they were ready to submit their final work, they would use the `git` commands.

Once the deadline passed, the instructor visited GitHub Classroom to review the code in each repository and leave comments for improvement.

C. Grading Assignments and Leaving Feedback

The instructor views the assignment in GitHub Classroom to see all the repositories that have been created by the students using the link provided in the assignment instructions. If the students make their own repository without using the link provided, the repository is not linked to the assignment. From there, the instructor goes through each file in the assignment. This is sometimes several files depending on the material covered in the assignment, like inheritance in Java, as an example. The instructor clicks on each line (or clicks and drags to select several) where they would like to provide feedback to reference it in a new *issue*. An issue can be used in industry by users to leave feedback or report bugs and can even be used as a way for an individual or group to assign tasks to be completed. An example of an issue can be seen in fig. 3. Issues typically cover one specific problem to fix.

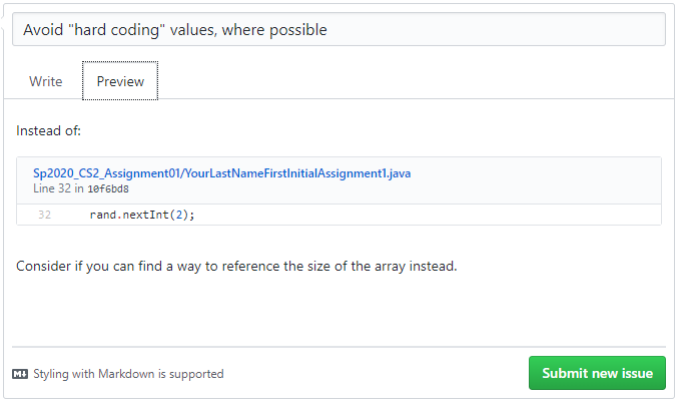


Fig. 3: Leaving Feedback as an Issue in GitHub

In addition to writing comments for a particular problem, a line of code can be referenced right in the issue, that when clicked on, will take the student directly to that line of code where it can be modified.

The editor used for issues (and pull requests) allows the use of Markdown, a language for formatting plaintext documents. With it, the instructor can add text formatted as code, tables, and images to the comments to provide additional detail that is easy to read and follow.

If the issue is assigned to a student, they are notified via email and can reply to continue the discussion or make changes to their code based on this feedback, prompting the student to place more emphasis on honing their skills than submitting something, receiving a grade, and not looking at it again.

To increase the speed of leaving feedback, when creating issues, the instructor compiled a list of frequently used comments from which to copy and paste into issues for other students, since students will largely make the same mistakes. This also allows the instructor to see a pattern in the comments left to clarify topics for which several students seem to be confused.

Once feedback was left in each student's repository, grades were assigned manually in the LMS. It is worth noting that GitHub Classroom has a way to connect with some popular

LMSes. The LMS used by this institution is not included in this list and we found it not easy to set up as it required several approvals from campus and technical administrators, so it was not used for this study.

IV. The Experience

There were some initial hurdles with getting the students started with `git`, which was to be expected. First, there was confusion on the basic command-line instructions for things such as navigating to a directory and whether the `git` commands were standard across operating systems. Two course sections were used for this experience and to distinguish between the two, we will refer to the course where `git` was used as the **experience section** and the other as the **base section**.

A. Submitting Assignments using `git` and GitHub

Concerning the `git` commands, the learning curve was much steeper than anticipated. Students were more focused on the required course work, so when it came time to submit the files in GitHub, they were at a loss, even with the video and written tutorials they could reference. In most cases, they ended up using GitHub's Add File drop-down menu to upload their file to the repository from their browser window rather than use a terminal window. Since the focus was on simply becoming familiar with the `git` commands and the process of using a VCS, this was acceptable. They were also given the option, after the first assignment, to use graphical interface tools for version control, namely SourceTree or GitHub Desktop, as they were are available for multiple operating systems. The students could use either of these products if they chose to, but no class time was dedicated to showing them how to use any tool in particular as the process for committing and pushing files to their repository, which we felt is the basis for how to use these tools, had previously been discussed. Furthermore, we wanted them to be able to get their feet wet, so to speak, and explore how to use these tools on their own.

The hope was that students would embrace the version control aspect of submitting their assignments with `git` and commit and push along the way; only about 19% of the students did. They started early and would commit as they made changes, leaving a full version history. The rest of the students worked as typical undergraduates, waiting until the last minute, pushing the final product.

B. Leaving Feedback in GitHub

At the time of writing this paper, leaving feedback in GitHub via issue tracking was more time consuming than we had hoped. The instructor of the course in our study used the emacs text editor in "org mode" [17] on a normal basis as a way to organize tasks and create to-do lists. It allows you to clock tasks in and out as you work on them to see how much time is being spent on a task. We used this feature to determine how long it took to leave feedback via GitHub, shown in Table I.

Each time the instructor stopped and restarted grading, a timestamp was recorded by the emacs text editor to give a duration for each grading session and a total was calculated for the task. It is of note that this method is contingent on the instructor remembering to clock in when starting and out when stopping (e.g., for a break, to teach a course, at the end of the day, etc.) and this was sometimes difficult. However, it does still provide a rough approximation of the time spent leaving feedback.

TABLE I: Duration for Leaving Feedback via GitHub and Annotated PDF

	Experience Section	**Base Section**
Assignment 1	4:40	0:47
Assignment 2	2:58	2:08
Assignment 3	3:50	2:42

Times are in hours:minutes format

For comparison, the same assignments were given the following semester in the base section, a CS2 course taught by the same instructor, containing 21 students, using the instructors usual way of leaving feedback via annotated PDFs. With roughly 30 of the 35 students submitting Assignment 1 in the experience section, it took a little more than 4.5 hours to leave feedback for each student, compared to 47 minutes for the 20 of 21 students in the base section.

We suspect this large difference in time could be due to several factors: the initial learning curve for the instructor in terms of leaving feedback in GitHub, the nature of leaving feedback via issue tracking, and the difficulty of grading the specific submissions. For the latter, it is our experience that students will often type their code at the last minute, with no time for testing and refining, leaving the instructor trying to determine how to assign a grade to the submission and with lots of feedback to give. This was quite a challenge with the issue tracker as it would redirect to a different page to create the new issue, forcing the instructor to navigate back to the home page of the repository after completing the feedback for that issue and start from the beginning, having to keep track of the last lines reviewed. This may only consist of a few mouse clicks, but was still tedious and time consuming. Additionally, the time to leave feedback for base section assignments does not include the time it took to download the files and convert to a PDF for annotation as both were negligible. The downloading process was relatively quick depending on network speed and size of assignments and the instructor previously wrote a python script to convert source code files to PDF. Another possible reason for the difference in time is that students rarely had the same problem, so more time than expected was spent typing different feedback for each student rather than being able to copy/paste feedback for a common problem.

The time was improved with Assignment 2 (only a 50 minute difference between experience and base), but increased again when giving feedback for Assignment 3. While there was less of a learning curve this time around, the assignment was the most complex of the three with multiple files that took longer to assemble into a single PDF per student for the base section and longer to leave feedback on for the experience section. The ability to create a feedback branch

and generate a pull request, as mentioned in [15], was added to GitHub Classroom between assignments 2 and 3 and it was used to give feedback for Assignment 3. This added a slight learning curve for the instructor once again, but proved to be a much faster and more streamlined way of both leaving and reviewing feedback. Comments were left with individual lines of code rather than in a separate window and the students were automatically notified when a code review was initiated via pull request. They could go to the pull request to view the feedback received and even reply to it, starting a conversation with the instructor, further enhancing their learning experience.

As a matter of fact, more students responded to the feedback left in GitHub, roughly 12%, than they did to feedback left in the LMS, roughly 10%. While this is not a very large difference, it is a step in the right direction.

Lastly, some of the students really embraced the version control process and committed files regularly and engaged in conversation on how to improve their skills. Others did not understand the underlying concept of version control and still submitted several versions of the same source files rather than reverting to a previous version and making modifications from there if necessary.

On the whole, it was a challenging, but rewarding experience that we would recommend to any instructors looking to add version control functionality to their programming classes.

C. Lessons Learned

There were quite a few things we learned during this experience, specifically:

- There should be an assignment for students to get acclimated with `git` and GitHub before starting their regular assignment submissions
- The instructor should use `git` during lecture while showing examples to get the students used to using it more often so they can see how it works and how it can be useful, as mentioned in [2]
- There should be some built-in commit dates throughout an assignment to motivate the students to start early and commit often so that there is a history of their work, allowing us to find patterns in how they program in order to better guide them
- With such a steep learning curve for the students to use the `git` commands, we should consider showing the easier ways to commit and push files to the remote repository up front or using a custom IDE as in [9]
- GitHub Classroom is improving, but still has a ways to go in terms of features to make it more classroom and assignment submission friendly. For example, students are allowed to push files to the repository after the deadline and it is not so obvious that the submission was late, so there needs to be a workaround for instructors when grading
- While issue tracking is a standard practice, it was very time consuming and seemed more cumbersome to allow for conversations, but pull requests excelled in this area
- We did not allow the students to merge changes made in response to feedback into their master copy, a standard part of code review in industry. This should be considered for future semesters or in more advanced classes
- The VCS environment is perfect for smaller, more frequent, incremental assignments that may be less stressful on instructor and student alike, allowing us to give more assignments and more feedback to improve students' skills

Note that the first two lessons are being implemented in a course that is ongoing at the time of writing this paper and they are proving to help the students get more quickly acclimated to the `git` commands and to committing their work often, leaving a history of their work for review.

V. Conclusion

In utilizing VCS in a small CS2 class, we encountered some resistance to `git`'s learning curve, which is steep even for professionals. Despite that, it was our experience that some students welcomed the idea.

Using these tools, we were able to provide detailed and meaningful feedback to students with a way to alert them that new feedback was available. We noticed that students would reply to the issues shortly after receiving the notification to justify their choices, ask for clarification, or express their newfound understanding of a concept. Some even went back to edit their code based on the feedback left in the issues, which we ordinarily would not be aware of unless they divulged this information.

We suspect that using this method to provide meaningful feedback to students will change their mindsets when it comes to completing programming assignments and improve their programming skills. To that end, our future plans include assessing how a change of this nature affects student learning, attitudes towards programming, and give them the tools they need to sharpen their skill set.

Our experiences using `git` for programming assignment submission and feedback thus far have been promising for the instructor. To ensure this is worth adding to courses throughout the curriculum, we need to get a student's point of view and that of other instructors in the discipline. Additionally, it would be of interest to see whether this could scale for large classes and/or those utilizing teaching assistants.

We believe doing so will allow students to learn about VCSs, which they would need for future employment. While it does not seem to reduce the time it takes to grade programming assignments, we remain confident it will help instructors in courses across the discipline improve their workflow and provide more substantial feedback in a way that encourages conversation, thus allowing us to further improve our students' skills.

References

[1] B. Cheang, A. Kurnia, A. Lim, and W.-C. Oon, "On automated grading of programming assignments in an academic institution," *Computers & Education*, vol. 41, no. 2, pp. 121–131, 2003.

[2] K. L. Reid and G. V. Wilson, "Learning by doing: introducing version control as a way to manage student assignments," in *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, 2005, pp. 272–276.

[3] L. Haaranen and T. Lehtinen, "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*. ACM, 2015, pp. 87–92.

[4] Github. [Online]. Available: https://github.com/

[5] K. Ala-Mutka and H.-M. Jarvinen, "Assessment process for programming assignments," in *IEEE International Conference on Advanced Learning Technologies, 2004. Proceedings.* IEEE, 2004, pp. 181–185.

[6] M. D. Adams, "Aristotle: A flexible open-source software toolkit for semi-automated marking of programming assignments," in *2017 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*. IEEE, 2017, pp. 1–6.

[7] [Online]. Available: https://classroom.github.com/

[8] S. Wasik, M. Antczak, J. Badura, A. Laskowski, and T. Sternal, "A survey on online judge systems and their applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, pp. 1–34, 2018.

[9] L. Yan, A. Hu, and C. Piech, "Pensieve: Feedback on coding process for novices," in *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019, pp. 253–259.

[10] J. Lawrance, S. Jung, and C. Wiseman, "Git on the cloud in the classroom," in *Proceeding of the 44th ACM technical symposium on Computer science education*. ACM, 2013, pp. 639–644.

[11] J. Kelleher, "Employing git in the classroom," in *2014 World Congress on Computer Applications and Information Systems (WCCAIS)*. IEEE, 2014, pp. 1–4.

[12] S. Gowtham, "Revision control system (rcs) in computational sciences and engineering curriculum." in *XSEDE*, 2014, pp. 76–1.

[13] C. Clifton, L. C. Kaczmarczyk, and M. Mrozek, "Subverting the fundamentals sequence: using version control to enhance course management," *ACM SIGCSE Bulletin*, vol. 39, no. 1, pp. 86–90, 2007.

[14] A. Zagalsky, J. Feliciano, M.-A. Storey, Y. Zhao, and W. Wang, "The emergence of github as a collaborative platform for education," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 2015, pp. 1906–1917.

[15] A. Rosenbloom, S. Sharmin, and A. Wang, "Git: Pedagogy, use and administration in undergraduate cs," in *Proceedings of the 2017 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, 2017, pp. 82–83.

[16] [Online]. Available: https://education.github.com/

[17] C. Dominik, *The Org Mode 7 Reference Manual-Organize your life with GNU Emacs*. Network Theory Ltd., 2010.