Adopting Git/Github within Teaching: A Survey of Tool Support

Richard Glassey
KTH: Royal Institute of Technology
Stockholm, Sweden
glassey@kth.se

ABSTRACT

The adoption and use of Git and Github within computer science education is growing in popularity. The motivation for this shift is strong: it combines a robust system for managing student coursework, sophisticated collaboration and communication tools for students and teaching staff, and an authentic experience of an important software engineering skill. Whilst previous literature has reported upon experience and benefits, there still exists a technical barrier to overcome in adopting Git and Github within an educational context. In response, both the community of teachers using Git/Github and the Github organisation itself have developed tool support to help solve the challenge of adoption, however these efforts are somewhat isolated and relatively unstudied. This work aims to provide an overview of these tools, identify the commonalities and differences, and develop a framework for comparison to assist teachers when looking for solutions for their own courses.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**;

KEYWORDS

Version Control System, Git/Github, Computing Education

ACM Reference Format:
Richard Glassey. 2019. Adopting Git/Github within Teaching: A Survey of Tool Support. In *ACM Global Computing Education Conference 2019 (CompEd '19), May 17–19, 2019, Chengdu, Sichuan, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3300115.3309518

1 INTRODUCTION

Version control systems (VCS) have become a vital component of managing resources and teamwork in modern software engineering [23]. This has led to an increasing demand upon teachers to prepare computing students for this future working environment by incorporating VCS into their courses. Software project courses that organise teams of students working together on a single project specification have been an initial point of adoption. However, use of VCS has spread further, for managing assignments, assessment and feedback within entry level CS1 courses and beyond.

CompEd '19, May 17–19, 2019, Chengdu, Sichuan, China

ACM ISBN 978-1-4503-6259-7/19/05…$15.00
https://doi.org/10.1145/3300115.3309518

There is a growing thread of literature within computing education that reports upon the benefits, challenges and experiences of adopting VCS into the learning environment. Examples can be found from the older centralised systems, such as CVS and Subversion [2, 21], to the more modern distributed and hosted systems, such as Mercurial and Git [17, 22]. In particular, Git combined with the hosting service Github has become the most popular approach, according to the literature [5, 9, 12, 15, 18]. Whilst this thread makes a convincing case for VCS in teaching, the problem still exists that this is not a system designed with education in mind. Barriers to adoption include (1) concerns about privacy and access control between students' work, (2) additional time it takes to investigate how VCS might work within a course, (3) complexity of using a system that is not designed to support teachers, (4) the uncertainty of how best to use VCS effectively for both students and teachers. As such, teachers have attempted to reduce these barriers by developing tools that integrate the common teaching tasks of distributing assignments and projects, managing and automating assessment, and providing feedback to students on their efforts. However, these efforts have been somewhat independent, isolated, and there has not been an effort to study their similarities and differences.

This paper has two main contributions. First, it provides a brief literature survey that charts the development of VCS usage within an educational context via different VCSs, with a focus on the use of Git/Github. Second, it surveys the open source solutions that teachers (and Github Inc.) have developed. The survey is structured according to the general aspects of each tool, their technical aspects, and most importantly their pedagogical features for supporting grouping of students, distribution of assignments, and support for assessment and feedback. Despite aiming at a similar problem, each of the tools have interesting differences, and bringing them together in this survey provides a useful reference for teachers who may wish to adopt such tools, or find inspiration for developing future tools that better meet the needs of teachers and students.

2 RELATED WORK

The use of VCSs in teaching has a history almost as long the systems themselves. One of the earliest mentions came in 1989, where students tasked with maintaining a Pascal compiler were disappointed *not to be able to use* the revison control system [3]. The importance of these change management and collaboration tools to software engineering is well established by now and is a defacto skill for all developers. This importance has motivated the adoption of such systems into teaching, either as a topic, or a means to manage the materials of a project-related course. As Git/Github continues to grow in popularity, more use cases in teaching are being revealed by practitioners and researchers. This section first develops the timeline of academic works that have reported the adoption and

experience of VCS, and then focuses specifically upon work that reports the adoption and experience of Git/Github.

2.1 Use of Version Control Systems in Teaching

Within software engineering courses, especially those oriented towards team projects, there is a strong desire to ensure that students make use of authentic tools in order to give them practical experience with the *training wheels removed*. Typically this implies the use of a serious integrated development environment (IDE) and a VCS. A series of articles proposed the potential value of using CVS within an educational context, however they did not report upon actual interventions with students [10, 19, 24].

In May 2003, teachers for a software engineering group project course at the University of Toronto adopted Concurrent Versions System (CVS) to manage the shared resources that students developed together for their coursework submissions [21]. Besides simplifying the coordination aspects between students (who had used email to share code previously), it was also reported as an effective way to teach version control principles in real time as the students used the tools on their own work. Teaching staff also benefited by being able to more easily share starter code in an immediately usable form (not zipped, archived or hosted on a web page), manage a student body spread over multiple campuses, and deliver feedback to students directly within the repository. Disadvantages included the risks of students damaging their repositories as well as preventing authorised access to the students' repositories.

As version control software evolved, from CVS to Subversion as the dominant choice amongst software engineers, teachers also moved with the changes in technology [2, 7]. In the Fall of 2005, teachers at the Rose-Hulman Institute of Technology (RHIT) adopted Subversion as a means to more effectively share the high volume of course resources between distributed staff, as well as also distributing and collecting student assignments [2]. The authors noted that this simplified the interaction with students, and allowed the direct and timely delivery of feedback, which was much faster than previous methods. From the teachers' perspective, this method also allowed them to spot-check students' work without having to explicitly request a submission, check the balance between group projects in terms of interaction counts, and detect when students were effectively (or not) distributing their work over time, and procrastinating before the deadline. The authors concluded that such an approach opened up a whole range of insights that were previously unavailable to them. By this stage, it was clear that version control, irrespective of tool, could find a useful place in support of teaching and learning, and general patterns for usage were suggested in the literature [20].

The centralised mode of revision control that CVS and Subversion used eventually became superseded by more flexible distributed version control systems (DVCS), such as Git and Mercurial. At the University of West Georgia, Mercurial was adopted as a DVCS for managing their CS1 course in Fall 2009 [22]. The authors noted that the complexity involved in using Mercurial was greatly reduced, compared to older centralised systems, like CVS and Subversion. This simplification encouraged the use of version control amongst students, and to such an extent that it was usable with entry level students in a CS1 course. They further identified that the administrative burden was also reduced when using DVCS, there was less chance of repositories being damaged by students, and there was better control over administrative access control to avoid students seeing other student's work. Most encouragingly, there was anecdotal evidence of students using Mercurial in courses that did not mandate its usage, indicating transfer of the skills.

2.2 Use of Git/Github in Teaching

Git was released in 2005, and is an example of a DVCS, which supports distributed and non-linear workflows. Github is both the organisation and the name of the web hosting service for Git repositories. Typically the two terms are rolled together or exchanged in common parlance. Whilst the previous section highlighted how CVS, Subversion and Mercurial have been used within educational contexts, there appears to be much more activity using Git and Github within educational contexts, as evidenced by the number of articles within the literature. There are two explanations for this: (1) Git/Github are the defacto VCS tool/platform for open source projects, and (2) Github Inc. have embraced the educational community by creating educational resources/training, hosting an online community, and providing free repository hosting for teachers.

One of the first reported uses of Git in a teaching context can be found in [17], as part of an operating systems course that made use of virtualisation and version control to manage student work, where similar benefits were found as per previous literature. In [18], the authors went beyond the normal discourse of describing the virtues of version control and focused instead upon the administrative benefits of using Git in coordination with hosting services that were cloud-based, rather than teachers needing to overcome the complexities of setting up local servers. As far as the literature is concerned, they were the first to report on their experiences of using Git with CS1 students. Another work reported some of the disadvantages with having dependencies upon external hosting services [1]. They used GitLab, which mirrors many features of Github, however due to the rapid pace of development, there were many issues, which resulted in negative feedback from students, with only some understanding the vision the teachers were trying to achieve. Nevertheless, the hosting service enabled advanced features, such as automatically triggering evaluation whenever a student pushed their work.

After these initial examples, further work elaborated upon the benefits, challenges and experiences of using Git and Github within a teaching context [5, 9, 12, 15]. In [15], one of the first to report within the literature on the availability of unlimited free repositories available via Github Inc. for teachers, noted the key benefits as being: (1) secure submission, (2) underwriting provenance and derivation of submitted work, (3) better source file organisation, (4) integrated Issue Tracking, and (5) exposure to standard industry practice. In contrast, [12] reported some of the underlying challenges that students faced, in spite of the many benefits reported in previous articles. In particular they highlight how students (1) may have had a badly formed mental model of version control, confusing branches with files and folders, (2) faced challenges with the terminology like `origin master`, (3) replaced communication via the issue tracker with physical communication whenever collocated with their teammates, and (4) missed out on an authentic

experience of VCS usage due to the changes required to manage distribution, assessment and feedback, which deviated from a regular software engineering project.

Despite the benefits and challenges that can be observed by teachers, the student voice has also been a source of insight into the experience of using Git/Github within courses. [9] reports that whilst the teachers were surprised by how overwhelming the student enthusiasm is for adopting VCS, they also discovered that they lacked understanding about the system or having confidence in their ability to use it effectively beyond the course. In [5], the benefits that students themselves reported were: (1) gaining and demonstrating industry relevant skills and practices, (2) enabling cross-team collaboration and contributions, (3) encouraging student contributions to course content, (4) breaking down the walled garden, and (5) version controlled assignments. Students also reported challenges that they encountered, including: (1) the perils of public projects, (2) unfamiliarity with Git and Github, (3) notification overload, and (4) Github is not designed for education.

More recently, attention has turned towards the value that can be derived from analysing the data that is generated when using a VCS. In particular, the Github API provides a wealth of information that can be used to generate input for learning analytics and educational data mining, as examples from software engineering projects have demonstrated [4, 8, 14]. In [13], the authors derived a range of indicators that could be used to more objectively and systematically determine the project quality and student behaviour in terms of individual and team contributions. These indicators then were found to be beneficial to encourage conformance within student teams to maintain quality both of their interactions as well as their behaviour within a software engineering course. A further example of possible indicators that teachers can use to improve their own courses was presented in [6], where the feedback captured in the issue tracker could be used to better understand individual assignments as well as differences in teaching assistant behaviour. In other works, automatic assessement has been added as a feature via the use of a continuous integration framework that runs testing automatically when students push their work, highlighting the extensibility of this platform[11, 16].

Despite the growth of interest in Git/Github by the academic community, one pressing issue is the technical barrier that must be overcome to make the most effective use of the technology. Besides the literature that has been presented here, various teachers and Github Inc. themselves have developed tools to support basic and advanced usage in terms of distribution of assignments, assessment and feedback. The next section surveys tools that have been shared as open source projects, with good documentation, which can support teachers who are interested in using this approach.

3 SURVEY OF OPEN SOURCE SOLUTIONS

The adoption of Git/Github introduces technical challenges to overcome. Besides the technical challenge in adopting Git/Github for teaching, there is a lack of information about solutions that may ease the burden, as well as no means to evaluate which may be the most suitable choice. Several teachers have shared their efforts online as open source projects, and Github Inc. has contributed to the community by providing official solutions as well. Undoubtedly, many more solutions have been developed by teachers working in isolation, however the scope here is limited to the tools that are publicly available as open source projects.

This section presents an analysis of the open source tools that were found. The following method was adopted. First, candidates for analysis were selected via keyword searching (keywords: *git | github | education | student | teacher*) of academic literature in Google Scholar, Github itself, and the education community discussion forum that Github Inc. has created[1]. Second, inclusion criteria for analysis required that the tool should be (1) publicly available as an open source project, and (2) well documented so that it is clear to understand the basic usage, main features, and central ideas behind the tool in its support for teachers and students. Third, a framework for evaluation was developed and applied that allowed different tools to be compared according to common criteria. To help organise understanding of the different tools, the criteria are organised into three sections that cover the general, technical and pedagogical aspects of each tool analysed.

3.1 General Aspects

The first component of the survey establishes some general aspects about each of the identified tools. Table 1 lists eight tools[2]. As might be expected, six of these tools originated from an educational context within a university. The geographic distribution is mostly tilted towards North America, but also includes two Scandinavian efforts. Github Inc. itself has created two tools: Teacher's Pet and Classrooms. Teacher's Pet was the forerunner of Classrooms, and as the last activity criteria shows, core development appears to have ceased in 2016. Interestingly, when searching for Teacher's Pet, it is clear that several people have forked the project and continued their own development of it, however these will be assumed to indistinct from the original for the purposes of clarity within this survey. All of the tools are hosted on Github itself and are open source projects, containing the source code, tutorial and documentation within each respective repository. For presentation convenience, the official Github URL shortener was used on each project repository URL[3]. At the time of writing, four of the projects displayed evidence of recent activity (as of October 2018), whereas four other projects showed no recent activity. It is unclear whether they had reached a level of maturity that change was no longer required, or that another solution had been adopted.

3.2 Technical Aspects

The second component of the survey switches attention to the technical aspects about each of the identified tools. Table 2 shows each of the tools in terms of their interface, implementation, and extensibility. In terms of interface, the majority of tools rely upon command line usage. Given the nature of regular and advanced Git interaction, this is not surprising. The teachers who adopt Git into their teaching most likely were already using it in their own practice. Whilst this is the obvious path, in terms of power and control that a command line interface affords, it also raises a technical

[1] https://education.github.community
[2] The author was also part of the Repomate project development, and efforts have been take to avoid deliberate bias or favour in the survey
[3] See: https://git.io/

Table 1: General aspects for evaluation of open source Git/Github tools

Tool	Organisation	Homepage	Last Activity
ACAD	UC Santa Barbara, USA	https://git.io/fxwj9	May 2014
Classrooms	Github Inc.	https://git.io/vGVC2	October 2018
GHClass	Duke University, USA	https://git.io/fxwjb	October 2018
Repomate	KTH: Royal Institute of Technology, Sweden	https://git.io/fxwjx	October 2018
Rhomboid	University of British Columbia, Canada	https://git.io/fxwjh	September 2017
Submit50	Harvard University, USA	https://git.io/fxree	October 2018
Teacher's Pet	Github Inc.	https://git.io/PCz4Vg	October 2016
Virtual Classroom	Simula School of Research and Innovation, Norway	https://git.io/fxref	October 2017

Table 2: Technical aspects for evaluation of open source Git/Github tools

Tool	Interface	Implementation	Extensibility
ACAD	Command Line	Python	N/A
Classrooms	Web Interface	Ruby	N/A
GHClass	Command Line	R	N/A
Repomate	Command Line	Python	Plugin Architecture
Rhomboid	Command Line	Python	N/A
Submit50	Simplified Command Line	Python	Internationalisation
Teacher's Pet	Command Line	Ruby	N/A
Virtual Classroom	Command Line	Python	How-to Documentation

barrier for many that may not be as comfortable with this form of interaction with a pedagogical tool. Two tools, Classrooms and Submit50, have taken a different approach. Classrooms simplifies the effort in creating repositories and distributing to students by making use of a web interface. Submit50 takes a different approach and simplifies the command line effort for students, by creating an abstraction over the regular git commands require to `add`, `commit` and `push` into a single `submit` command.

In terms of implementation, Python is the most dominant programming language used for development, with five tools making use of it. Ruby is used in both of the Github Inc. solutions, due to Ocktokit, the official wrapper for the Github API being implemented in Ruby. Demonstrating the flexibility of developing against the Github API, one tool is implemented in R, due to the data science context of its usage. It is worth noting that wrappers for the Github API have been developed and a list is maintained by Github[4].

Each of the tools supports a range of basic features, as described in Sec. 1. However, as each educational context may have different requirements beyond the basic use cases, the final criteria considers the support for custom extensions. Only three projects included explicit support for extensibility. To be clear, all of the projects are open source and by default can be forked and extended endlessly, but here we are most interested in tools that have the potential to adapt to new use cases with minimal effort or grow through 3rd party contributions. Therefore, tools that did not make explicit mention of how to extend their work are marked with N/A.

Repomate was the only project to anticipate this need, and has a plugin architecture that allows teachers to develop extensions depending on their use cases. For example, three core plugins included as exemplars integrate the actions of compiling code (via `javac`), running unit tests (via `junit`), and linting code via static analysis (using `pylint`). Submit50, whilst not providing explicit documentation on adding new features, takes a different approach to extensibility and includes the ability to internationalise the tool. This is a nice addition that allows reuse in many different geographic contexts. Finally, Virtual Classrooms includes guides within their documentation that shows where in the source code that extension can be made, in order to alter the behaviour of the tool.

3.3 Pedagogical Aspects

The third and final component of the survey considers the pedagogical aspects of each of the tools. This is by far the most important aspect of this analysis, and the findings are shown in Table 3. The first two survey aspects contribute which tools exist, as well as establishing some relevant technical information. The pedagogical aspect attempts to dive into the ways that Git/Github have been used to support education, and then use these as a means of differentiation between the tools presented here. Four key criteria are considered: which grouping of students is supported, the model

[4]https://developer.github.com/v3/libraries/

Table 3: Pedagogical aspects for evaluation of open source Git/Github tools. *ORpSpA* stands for one repository per student per assignment. *OBpP* stands for one branch per problem, which uses a single repository with multiple branches.

Tool	Grouping	Distribution	Assessment	Feedback
ACAD	Solo & Pair	ORpSpA	Batch clone	N/A
Classrooms	Solo & Team	ORpSpA	Batch clone — requires additional application	Travis CI Support
GHClass	Solo & Team	ORpSpA	N/A	Wercker Badge Support
Repomate	Solo	ORpSpA	Batch clone, Plugins, Peer Assessment	Batch open/close issues
Rhomboid	Solo	ORpSpA	Grading & Peer Assessment	Integrated Rubric Feedback
Submit50	Solo	OBpP	Grading — requires Check50 tool	N/A — requires Check50 tool
Teacher's Pet	Solo	ORpSpA	N/A	N/A
Virtual Classroom	Solo	ORpSpA	Peer Assessment	N/A

of repository distribution, support for assessment, and support for feedback to students.

3.3.1 Grouping. Git/Github was not designed as a learning management system (LMS), but this has not stopped inspired teachers from finding ways to retrofit important aspects that a traditional LMS would support. Key amongst these is how to arrange students and their assessments, answering the question of whether this should be solo work or team work.

All tools supported solo work, where an individual student works in isolation, and others cannot view their work. At first glance, an open source platform may seem to be at odds with this use case. Whilst students could create their own private repositories, this does not help as most often each tool is trying to help a teacher distribute an assignment to students, and relieve them of the burden of creating a repository correctly that can then be accessed and assessed by a teacher at a later date. The common solution adopted by the tools considered here is to create an organisation on Github to establish ownership rights of all repositories, then, to prevent students from seeing other students work, create teams that only contain a single student.

With this model of creating teams, it is then a matter of adding new members to support groups of students, whether for group project work (e.g. Classrooms and GHClass), or for peer assessment (e.g. Rhomboid). ACAD was the only project that differed in that grouping could be either solo or pairs, but none of the documentation indicated that it was flexible in raising the size of group. Classrooms differed from GHClass in that students were expected to find their way to the correct team via the web interface. GHClass notes in their documentation that most educators would prefer to have this control themselves and create predefined teams rather than rely on students to get it right.

3.3.2 Distribution Model. One of the driving motivations to create these tools in the first place is to generate and distribute assignments to a class of students. As classes can become large, achieving this manually is not realistically possible. However, each tool at a basic level works on the notion of taking as input a roster of students, and then as output, assignments that students can work on within the Git/Github environment. Two models of how to manage this have emerged from the analysis of these tools. The dominant model is *one repository per student per assignment* (ORpSpA), noting that we can substitute student for team depending upon the desired grouping. The alternative model that is employed in only one tool considered here is *one branch per problem* (OBpP).

For ORpSpA, this is the most simple solution to implement. For every given student, create a repository with identical contents to a master repository containing the assignment materials for a specific assignment. Whilst seven of the tools considered here use this model, they implement it in different ways, but the result is the same. Typically the student will see a repository in their Github account that is named with the following pattern, or similar derivation: `<student-name>-<assignment-number/name>`.

On the other hand OBpP uses an alternative model, where the student has one repository, but many branches within that repository to store the solutions for their assignments. Whilst this is slightly more complex to implement, requiring a deeper understanding of Git branching, it does create some nice advantages. First, the student is not overwhelmed by having many repositories. Second, teaching staff are not overwhelmed by having N students × M assignments to keep track of. Finally, students are exposed to branching early (although perhaps not quite how it is supposed to be used) and get to make use of pull requests to signal that they are ready to have this particular branch/problem assessed.

3.3.3 Assessment. In terms of support for assessment, differences emerge in the level of sophistication used by each of the tools, influenced by the workflow that each tool follows. In some tools, there is no support for assessment, and they only help with the task of distributing assignments to students. Others provide limited support to help with the task of gathering student assignments for assessment via batch cloning. Finally, some tools provide a much more automated workflow with integrated support for assessment.

Teacher's Pet and GHClass do not explicitly mention support for assessment. ACAD, Classrooms and Repomate all provide different solutions for the task of batch-cloning student repositories to a local file-system in order to conduct assessment. Batching cloning gives the teacher the ability to clone all repositories for a particular assignment (Repomate also supports sub-groups of students for multiple teaching assistant use case). Classrooms differs the most because it makes use of a stand-alone desktop application to handle

the bulk cloning of repositories, whereas both ACAD and Repomate use the command line interface consistently for all operations. Repomate also integrates the use of plugins that execute third party tools (e.g. `javac, junit, pylint`) once a batch of repositories has been cloned. A typical use case of this would be to clone all repositories for a group of students, locate and run all unit tests in each repository, and generate a report of the outcome.

Submit50 can be combined with a separate tool, Check50, that allows both student and teacher to run assessment operations automatically upon work that has been committed to a repository. This creates a more symmetric level of access and heightened interactivity with the system for students, which is in contrast to other approaches that are asymmetric in the sense that students submit and then are forced to wait until teachers have assessed their work and communicated the results.

Rhomboid has the most sophisticated assessment system that is entirely integrated into Git/Github. First, it supports the notion of open and closing assignments, which alters the visibility for students. Second, it connects teaching assistants with the students' work they have to assess, and integrates the grading rubric and results into special repositories for tracking results. Third, once teaching assistants have graded students, the students can then update their own repository and see their results. Fourth, this builds up over all assignments and can deliver a grade profile and final result for the course. By far this was the most advanced system used that only depends upon the tool and Github itself.

Finally, Repomate and Virtual Classroom both support peer assessment. This takes advantage of the underlying team model for organising repositories for individuals, then this allows additional students to be added at a later date, with the intention that they will perform a peer review of the individual work.

3.3.4 Feedback. As Github has good support for communication and coordination between developers in terms of the issue tracker and the pull request mechanisms, a rich opportunity for student feedback presents itself. Once more, each of the tools adopted different strategies. ACAD and Teacher's Pet did not have explicit support for feedback. Virtual Classrooms does not have built in support for feedback, but presumably the peer assessment relies upon the use of issue tracking to capture the peer feedback. Submit50 itself does not have explicit support for feedback, but as mentioned in the previous section, Check50 will assess the student work automatically and will also generate feedback, however this is received via the command line, rather than via the issue tracker. Rhomboid is similar, but updates the readme file that sits at the root of the repository. Repomate differs by supporting the bulk opening and closing of issues on the issue tracker. Here the two main use cases were to take the output of the plugins (e.g. `junit`) and post this to the issue tracker for a specific student, or to post course wide updates and grading criteria in the form of an issue.

Finally, both Classrooms and GHClass connect with continuous integration (CI) services in order to generate feedback for student assignments whenever they push their attempts. In the case of Classrooms, Github Inc. has an agreement with Travis CI that allows educational users to have their repositories connected to a CI Server, as long as a simple configuration file is included within the student repository. The typical use case for a teacher would be to have a unit test suite setup that the student must pass before receiving their final grade. For GHClass, another CI service, Werker is used. It was not clear in the documentation quite how this was meant to be used, but there was evidence of using status badges on a repository in order to convey some status that may be helpful for the student. Typically these badges are found on open source projects to indicate whether tests are passing and so on.

3.4 Limitations of Survey

This survey was a first attempt to bring together work that has focused on a common problem. Part of the value is formulating a basis for comparison, then stimulating further discussion of these ideas so more teachers can benefit. However, there are some important limitations to report on this effort:

(1) Data was sourced from public material — Features may have been missed or used alternative terminology. Future work will survey teachers that have created similar tools.
(2) Other languages have not been considered - As reflected in the North American bias, other international tools might have been missed.
(3) Forked versions of projects (e.g. Teacher's Pet) not considered — Extensive modifications could have been made to projects that could reveal solutions to criteria deemed N/A. The main justification here was that documentation was not available to make an informed decision.

4 CONCLUSION

There is a growing thread of research on the use of VCS in educational contexts. The benefits, challenges and experiences have been shared across the computing education community. Whilst promising, this work has been somewhat independent and isolated, and there is an ongoing need to keep track of developments and ensure that knowledge is gathered and shared effectively.

The survey of eight tools presented here considered their general, technical and pedagogical aspects. This is only scratching the surface, constrained by what is publicly discoverable — many other teachers may have created solutions to the tasks of grouping students, distributing assignments, and managing assessment and feedback, but not shared them publically. However, as this survey has illustrated, there are commonalities and differences in the approach to supporitng these tasks. By bringing these tools together in this survey, it is hoped that (1) teachers will not waste time by starting from scratch, and (2) new tools and innovative features can build upon the knowledge gathered.

What is most promising is the ongoing development in this area. Half of of the tools surveyed are still active at the time of writing, and furthermore, Github Inc. has devoted resources towards supporting teachers in their exploration of using VCS by both providing free resources, tools, and a community for teachers with similar interest to share their knowledge. In terms of what comes next, the future seems bright. The literature review and tools surveyed provide evidence that this area is growing. Whether Git/Github remains the defacto approach remains to be seen. However, the integration into modern teaching environments for both students and teachers have had a clear impact already and will continue to do so within computing education.

REFERENCES

[1] Miroslav Biňas. 2013. Version Control System in CS1 Course: Practical Experience. In *IEEE 11th International Conference on Emerging eLearning Technologies and Applications (ICETA'13)*. pp. 23–28. https://doi.org/10.1109/ICETA.2013.6674398

[2] Curtis Clifton, Lisa C Kaczmarczyk, and Michael Mrozek. 2007. Subverting the Fundamentals Sequence: Using Version Control to Enhance Course Management. In *Proceedings of the 38th ACM Technical Symposium on Computer Science Education (SIGCSE'07)*. ACM New York, NY, USA, pp. 86–90. https://doi.org/10.1145/1227310.1227344

[3] B J Cornelius, M Munro, and D J Robson. 1989. An approach to software maintenance education. *Software Engineering Journal* 4, July (1989), pp. 233–236. https://doi.org/10.1049/sej.1989.0030

[4] Valerio Cosentino, Javier Luis, and Jordi Cabot. 2016. Findings from GitHub: Methods, Datasets and Limitations. In *Proceedings of the 13th International Workshop on Mining Software Repositories (MSR'16)*. ACM New York, NY, pp. 137–141. https://doi.org/10.1145/2901739.2901776

[5] Joseph Feliciano, Margaret-Anne Storey, and Alexey Zagalsky. 2016. Student experiences using GitHub in software engineering courses: a case study. In *Proceedings of the 38th International Conference on Software Engineering Companion (ICSE'16)*. 422–431. https://doi.org/10.1145/2889160.2889195

[6] Richard Glassey. 2018. Managing assignment feedback via issue tracking. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE 2018)*. ACM New York, NY, USA, p. 382. https://doi.org/10.1145/3197091.3205819

[7] Louis Glassy. 2006. Using version control to observe student software development processes. *Journal of Computing Sciences in Colleges* 21, 3 (2006), pp. 99–106.

[8] Georgios Gousios and Diomidis Spinellis. 2017. Mining Software Engineering Data from GitHub. In *Proceedings of the 39th International Conference on Software Engineering Companion (ICSE-C '17)*. IEEE Press, Piscataway, NJ, USA, 501–502. https://doi.org/10.1109/ICSE-C.2017.164

[9] Lassi Haaranen and Teemu Lehtinen. 2015. Teaching Git on the Side. In *Proceedings of the 20th Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'15)*. ACM New York, USA, 87–92. https://doi.org/10.1145/2729094.2742608

[10] Ken T. N. Hartness. 2006. Eclipse and CVS for Group Projects. *Journal of Computing Sciences in Colleges* 21, 4 (2006), pp. 217–222.

[11] Sarah Heckman and Jason King. 2018. Developing Software Engineering Skills using Real Tools for Automated Grading. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 794–799. https://doi.org/10.1145/3159450.3159595

[12] Ville Isomöttönen and Michael Cochez. 2014. Challenges and Confusions in Learning Version Control with Git. In *Communications in Computer and Information Science*, Zholtkevych G. (eds) Ermolayev V., Mayr H., Nikitchenko M., Spivakovsky A. (Ed.). Vol. 469. 178–193. https://doi.org/10.1007/978-3-319-13206-8_9

[13] An Ju and Armando Fox. 2018. TEAMSCOPE: Measuring Software Engineering Processes with Teamwork Telemetry. In *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education (ITiCSE'18)*. ACM New York, NY, USA, pp. 123–128. https://doi.org/10.1145/3197091.3197107

[14] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016), pp. 2035–2071. https://doi.org/10.1007/s10664-015-9393-5

[15] John Kelleher. 2014. Employing git in the classroom. In *World Congress on Computer Applications and Information Systems (WCCAIS'14)*. IEEE, pp. 1–4. https://doi.org/10.1109/WCCAIS.2014.6916568

[16] Stephan Krusche and Andreas Seitz. 2018. ArTEMiS: An Automatic Assessment Management System for Interactive Learning. In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*. ACM, 284–289. https://doi.org/10.1145/3159450.3159602

[17] Oren Laadan, Jason Nieh, and Nicolas Viennot. 2010. Teaching Operating Systems Using Virtual Appliances and Distributed Version Control. In *Proceedings of the 41st ACM technical symposium on Computer Science Education (SIGCSE'10)*. ACM New York, NY, USA, pp. 480–484. https://doi.org/10.1145/1734263.1734427

[18] Joseph Lawrance, S Jung, and Charles Wiseman. 2013. Git on the cloud in the classroom. In *Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE'13)*. pp. 639–644. https://doi.org/10.1145/2445196.2445386

[19] Stephen Paul Linder, David Abbott, and Michael J. Fromberger. 2006. An instructional scaffolding approach to teaching software design. *Journal of Computing Sciences in Colleges* 21, 6 (2006), pp. 238–250.

[20] Ivan Milentijevic, Vladimir Ciric, and Oliver Vojinovic. 2008. Version control in project-based learning. *Computers and Education* 50, 4 (2008), pp. 1331–1338. https://doi.org/10.1016/j.compedu.2006.12.010

[21] Karen L. Reid and Gregory V. Wilson. 2005. Learning by Doing : Introducing Version Control as a Way to Manage Student Assignments. In *Proceedings of the 36th ACM technical symposium on Computer Science Education (SIGCSE'05)*. ACM New York, NY, USA, pp. 272–276. https://doi.org/10.1145/1047124.1047441

[22] Daniel Rocco and Will Lloyd. 2011. Distributed version control in the classroom. In *Proceedings of the 42nd ACM technical symposium on Computer Science Education (SIGCSE'11)*. pp. 637–642. https://doi.org/10.1145/1953163.1953342

[23] Ian Sommerville. 2015. *Software Engineering* (10th ed.). Addison-Wesley.

[24] Ying Liu. 2004. CVS historical information to understand how students develop software. In *International Workshop on Mining Software Repositories (MSR'04). In proceedings of the 26th International Conference on Software Engineering (ICSE'04)*. ACM Press, pp. 32–36. https://doi.org/10.1049/ic:20040472