Scaffolding Version Control into the Computer Science Curriculum

Denise M. Case, Nathan W. Eloe
School of Computer Science and Information Systems
Northwest Missouri State University
Maryville, MO 64468, USA
{dcase,nathane}@nwmissouri.edu

Jennifer L. Leopold
Department of Computer Science
Missouri University of Science and Technology
Rolla, MO 64468, USA
leopoldj@mst.edu

Abstract

Version control systems (VCS) are widely-used in the software industry. They provide a powerful, collaborative framework that allows software engineers to work together effectively. VCS allow users to track changes and merge ongoing work into concurrently evolving software projects. Distributed VCS such as Git, allow a great degree of flexibility, and provide powerful options for managing personal code and evolving collaborative content. Power incurs responsibility, and introducing collaborative coding and version control tools to new developers can create many challenges. Yet these tools, once mastered, are crucial skills for professional developers. In this paper, the authors introduce VCS to computer science students both in a custom environment specifically designed to support new developers and in a commercially-available native environment suitable for more experienced students. Results show that proper introduction of these powerful tools can make early exposure a positive and valued experience.

Keywords. *collaborative systems, computer science education, education technology, pedagogy, profession-based learning, scaffolding, version control, visual language*

1 Introduction

Computer science courses aim to prepare students to think computationally and *apply* their thinking in a practical way. Many powerful tools and techniques are being developed in both academia and industry to assist with the practical application of computer science and software engineering. Incorporating a variety of these useful elements into a crowded curriculum can be challenging, but also increasingly valuable, both for the benefits it offers graduates in early employment, as well as in the way these tools and techniques can be used to enhance the immediate educational experience. One tool that has developed to be nearly ubiquitous in industry is the application of advanced *version control systems* (VCS). In industry and in its prequel, *project-based learning*, development projects require collaboration between teams of developers to effectively and efficiently create software that addresses real-world needs. Application of VCS and the associated mastery of these systems facilitates teamwork, collaboration, and effective and enjoyable creation of significant and useful software.

VCS and other professional tools are often introduced later in the educational curriculum, possibly in a software engineering course, generally a second-year or later course, and may not be reinforced through repetition and application in later classes. This is unfortunate, because powerful tools such as VCS can also be difficult for new users and especially those new to programming and software in general. The power and flexibility of new distributed VCS can make them especially challenging, unfriendly, and distracting for students already fully engaged in developing core competencies required in the computer science curriculum.

This paper, in conjunction with concurrent research [1], is an investigation into better approaches to introduce version control into pedagogy. For novice, less experienced students, we propose introducing these powerful version control tools by providing facilitative scaffolding in the form of a simple, consistent, visual language, that supports students during their introduction and initial exposure to VCS. For more advanced students, who may still be entirely or mostly unfamiliar with VCS, we propose a series of short, simple tasks to progressively integrate version control while working in a native environment.

2 Background and Related Work

In this paper, we build on research in instructional technology and pedagogy, in visual languages, in collaborative frameworks, systems, and learning, and on prior work on

DOI reference number: 10.18293/DMS2016-036

the introduction of version control systems in the computer science curriculum.

2.1 Instructional Technology and Pedagogy

Software engineering and development can be challenging; a variety of pedagogical approaches have been developed to assist with the learning process [2]. One popular technique is called *scaffolding*, designed specifically to support novices as they increasingly master challenging tasks [3]. As the novice gains proficiency, scaffolding supports are progressively removed through a process called *fading*. This continues until the supports are no longer needed. In this paper, we propose approaches for assisting students as they gain exposure and mastery with VCS and propose two approaches that allow VCS to be efficiently and effectively integrated into the learning process.

2.2 Visual Languages

Visual languages continue to grow in importance; the success of visual languages is evident in the proliferation of iconic-based languages on smart phones [4]. Such systems need to be able to reach non-expert users as intuitively as possible, and as multimedia applications continue to develop and expand, we expect visual language systems, both special- and general-purpose, to continue to grow in importance, both as the subjects of theoretical research and in actual practice [5].

2.3 Collaborative Frameworks, Systems, and Learning

Current trends in education include the integration of online, hybrid, and collaborative learning environments [6, 7]. Modern collaborative environments in higher education are responding to many of the same opportunities as industry, including the increasing application of smart technologies, such as cloud computing for software distribution and usage and the adoption of collaborative paradigms and environments that leverage the work of student teams just as they do software development teams in industry [6]. In profession-based learning and outcomes-based education, collaboration is both a teaching strategy and a learning outcome; in both public institutions and large organizations, the ability to work together is critical to successful software engineering [8, 9]. The application of Computer Supported Collaborative Learning (CSCL) to software engineering has been the subject of recent research, including versioning systems with positive results [10].

2.4 Version Control in Education

For over ten years, educators have employed a variety of VCS to promote collaboration and support student engagement in the classroom [11] and Git, specifically, has been the focus of current research and used to help students work collaboratively and receive instructor feedback [12, 13].

Widely-employed in professional software development, these powerful systems offer considerable benefits to educators, but adoption continues to suffer from the perception that VCS are difficult, time-consuming, or could distract from the core content of the course [14, 15]. Recent research recommends VCS integration *throughout the curriculum*, in a *continuous and gradual manner* [15].

Hosting solutions such as Bitbucket and GitHub offer special packages for students that include unlimited free private repositories and other benefits. To support educational users, GitHub created GitHub Classroom, a set of utilities on top of its hosting service that automates the creation of repositories and associated permissions. GitHub Classroom supports teachers by allowing projects to be assigned in various ways with various degrees of visibility to other students in the class. It integrates with Continuous Integration tools, allowing unit tests to be run on assignment submission and enables *auto-grading* in a consistent manner. Use of Bitbucket, GitHub, and GitHub Classroom requires students to create and use their own student accounts.

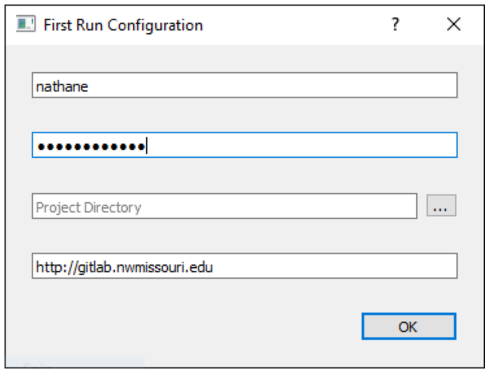
3 GitSubmit: A Custom Environment for Introducing VCS to Novice Developers

GitHub Education and GitHub Classroom are powerful, well-designed tools that enable the instructor to automate the creation of assignment repositories and provide feedback to students. However, these tools do not simplify students' interaction with the submission system; students must use their tool of choice to interact with the system. This can be overwhelming to students just learning to program and develop software. Developing software to solve problems requires a form of thinking foreign to many students of programming, and they are constantly being introduced to new tools such as IDEs and compilers. Submission systems designed for other kinds of assignments (such as essays or worksheets) are not ideal, but are used because they are in place and familiar to the students.

GitSubmit is an attempt to provide an interface to students that is simple, requires little to no interaction from an "administrative" standpoint, and is secure (and preserves academic integrity). Unless a student wants to further interact with GitLab, the only required interaction is changing their account password from the default. GitSubmit is built around a self-hosted GitLab [16], an open source Git hosting solution that allows accounts to be created with the students' campus usernames during course creation.

3.1 Student Interaction with GitSubmit

GitSubmit does not require the student to install any software (including Git); it is a packaged Python executable that contains all required functionality, including ssh, authentication, and Git commands. After being greeted on first load by a simple configuration window (Figure 1) where they specify their username, password, and a place to put all of the assignments it checks out, the student sees the assignment window as shown in Figure 2. The students can select an assignment to work on by selecting the semester and class from the tree-based menu on the left hand side of the application. They begin a project by selecting the assignment, then clicking the button circled in Figure 2 (which maps to a combination of `git pull` and `git clone`, depending on whether the assignment has already been started). Upon doing so, they see the project has been downloaded as in Figure 3.


First Run Configuration

nathane

•••••••••••••

Project Directory

http://gitlab.nwmissouri.edu

Figure 1: Configuring Gitsubmit

A student working on a project writes their code in the project directory, selects modified and new files in the "Unstaged Changes" window, then clicks the arrow between the Unstaged and Staged Changes windows (see Figure 4). This button maps to `git add`. After all required changes are added to the Staged Changes pane, the student writes a brief commit message in the box, then selects the right-most arrow (`git commit`) to commit their code. At this point, the student sees something similar to Figure 5; all that remains is to push the upload button (circled in Figure 5), which will perform a `git push` operation, and sync the code to the central repository. The green checkmark icon in the commit window will appear next to the top-most commit, signifying that it is the version of the code that will be accessed by the grader (or whoever does the next `pull`, such as a partner).

To improve security and streamline the submission process, the student's password is not saved; it is used only once when the program is first run. The password is used to obtain a token that allows the interface to authenticate to the GitLab API as the user without a password. Additionally, when a student logs in for a first time from a computer, a new SSH key is generated and used for all Git operations, specifically `push`, `pull`, and `clone`. In this way, no passwords are saved, and if the student loses their computer, access can be revoked by removing the SSH key from their account.

3.2 Instructor/TA Interaction with GitSubmit

Just as the students have a simplified interface for interacting with the submission system, the instructor interface is designed to automate much of the tediousness behind creating classes and assignments. The instructors, TAs, and graders are required to install Git and have it on their `PATH` so the interface can access it; this is because there are certain advanced features that are much more reliable using the standard Git client. The added setup complexity of the instructor's interface is worth it for the powerful features GitSubmit provides.

When the teacher wants to start a new class, they open the instructor interface, where they must provide information such as the semester, year, and a CSV containing a course roster. The instructor interface uses the CSV to ensure that all students have accounts on the system; if a student in the class does not exist in GitLab, the student's user is created using their email address as their username and a default password. The interface creates a GitLab group with teacher as the owner, and a specially formatted group name that will be used to populate the lists of courses and semesters for both instructors and students. Students are *not* added to this group, as it would allow them to view each other's assignments, but graders and TAs are added to the group so they have access to every students' submissions. The course roster is stored as a Git repository within the group.

To add an assignment, the teacher navigates to the "Add Assignment" tab, selects the course from the list (which is populated by using the list of groups the teacher is an owner of), provides an assignment name, and either a project description (in Markdown), or a skeleton directory (which is useful for distributing tests or boilerplate code). Optionally, they may also provide a comma-separated values (CSV) file containing groups. The interface creates a repository for each user (or group) with the name format `<lname><finitial>_<assignment_name>` (example: `eloen_lab11`) on the GitLab server, adds the student as a developer on the repository (so they do not have permission to change visibility to other students), and

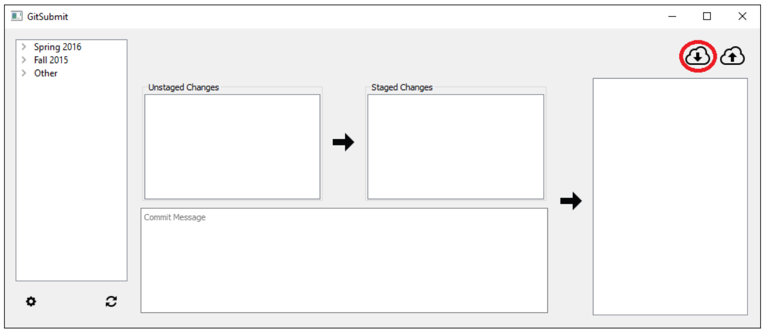
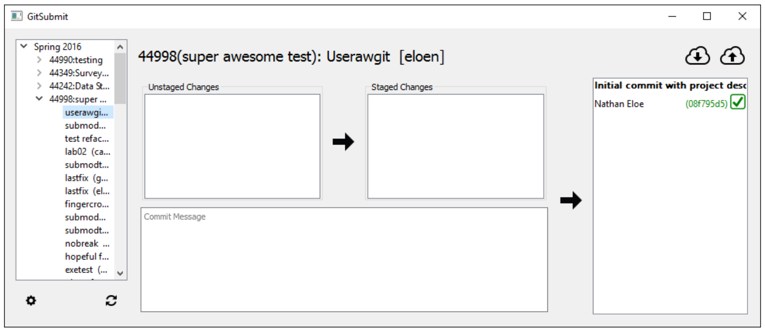


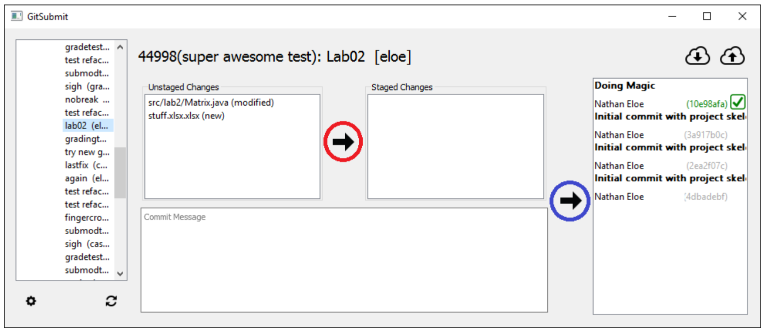
Figure 2: The Gitsubmit main window (project download button circled in red)



44998(super awesome test): Userawgit [eloen]

Initial commit with project desc
Nathan Eloe (08f795d5) ✓

Figure 3: A successfully downloaded project

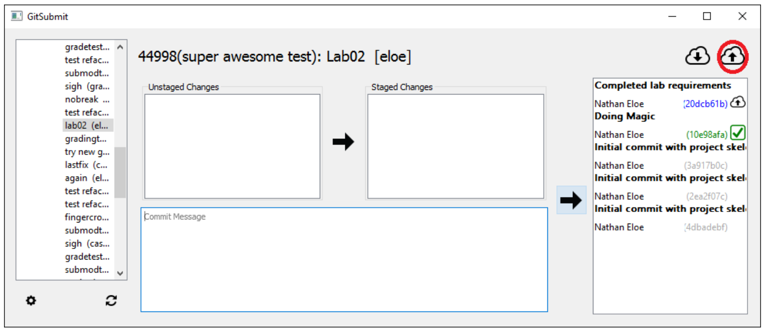


44998(super awesome test): Lab02 [eloe]

src/lab2/Matrix.java (modified)
stuff.xlsx.xlsx (new)

Doing Magic
Nathan Eloe (10e98afa) ✓
Initial commit with project skel
Nathan Eloe (3a917b0c)
Initial commit with project skel
Nathan Eloe (2ea2f07c)
Initial commit with project skel
Nathan Eloe (4dbadebf)

Figure 4: Committing your code.



44998(super awesome test): Lab02 [eloe]

```
gradetest...
test refac...
submodt...
sigh (gra...
nobreak ...
test refac...
lab02 (el...
gradingt...
try new g...
lastfix (c...
again (el...
test refac...
test refac...
fingercro...
submodt...
sigh (cas...
gradetest...
submodt...
```

```
Completed lab requirements
Nathan Eloe          (20dcb61b)
Doing Magic
Nathan Eloe          (10e98afa)
Initial commit with project skel
Nathan Eloe          (3a917b0c)
Initial commit with project skel
Nathan Eloe          (2ea2f07c)
Initial commit with project skel
Nathan Eloe          (4dbadebf)
```

Figure 5: Finishing up.

pushes the base repository (skeleton directory or description `.md`) to GitLab. Additionally, the interface creates a single repository containing all student assignment repositories as Git submodules (for more information on Git submodules, see Chapter 7 Section 11 in [17]). This enables whoever is grading an assignment to clone a single repository, then get the newest version of all submissions by running `git submodule update --init --recursive; git submodule foreach git pull origin master`.

3.3 Submission Workflow in GitSubmit

The interactions above correspond directly to the normal academic submission process; to assign work to students, a teacher must write the assignment description, distribute it to students, and collect and grade the final submissions. Using GitSubmit, the instructor formats the assignment description (or project skeleton), and distributes it to students using the "Add Assignment" functionality of the instructor interface. Students then use the GitSubmit student interface to retrieve the assignment description and complete the assignment. A student can then submit his or her work by pushing to the Gitlab server (most submissions require either uploading to a digital dropbox or handing physical paper to the instructor). Collection of the submissions is done using the `git submodule` commands to pull the latest versions of each student's work. The individual grading the work can then grade as they normally would. Feedback beyond a final grade can be provided to students using Gitlab's comment infrastructure, which allows people to leave comments and questions associated with individual lines of code (a level of clarity one can not always achieve when using traditional learning management system (LMS) submission capabilities).

GitSubmit has the potential to vastly streamline the process of assignment submission beyond what has been implemented thus far. Because collecting submissions can be done with two simple commands, enforcing strict deadlines can be done using a task scheduler or a cron job. Additionally, instructors can take advantage of Gitlab's webhooks (and Git's pre- and post-commit hooks) to automate objective evaluation of assignments using a Continuous Integration (CI) server. Automated application of CI, static code analysis tools, and automated testing can help identify incorrectness and assess a variety of "soft" programming elements, such as adherence to coding standards. Any time savings due to such automation may allow graders to spend more time reviewing submissions at a higher level of judgement and which allows graders to spend more time evaluating student submissions on higher levels of code quality and enable teachers and teaching assistants to spend more time identifying areas for individual assistance and working with struggling students.

4 Native Git: A Gentle Introduction for Upper Level Courses

Higher-level students may be several years past their bachelor's degrees with several years of work experience. However, it is not unusual for these students to primarily be working as independent developers and have little or no experience with version control systems.

The goal for these more independent and experienced students is to introduce them to version control in a positive way and encourage them to incorporate it into their personal workflow. If we are successful, the hope is that the students will independently choose native solutions for subsequent projects and be better prepared for industry opportunities that involve coding on a professional team.

To this end, Git was introduced over a series of assignments by integrating version control into a series of assignments in an upper-level course focusing on ASP.NET Core. Bitbucket was selected as the native hosting platform because its student program at the time was the only one to offer free, unlimited private repositories for students and faculty, a feature the authors have greatly appreciated and employed.

4.1 Getting Started

The initial assignment was conducted in a shared discussion forum, *Get Started with Bitbucket*.

1. Go to the Bitbucket Cloud Documentation Home [19].
2. Under *Get started*, click *Create an account* and follow the steps to create a personal account with your university email.
3. Under *Get started*, click *Set up version control* and follow the steps to *Set up Git* and *Install Git for Windows* (use the installation defaults). In Step 4, configure your user.name and in Step 5, configure your email.
4. You may also do Step 2 if you like; it is helpful, but not required.
5. When successful, reply with a subject of *Team x - Your Name - Success* and how long the assignment took.
6. Include a screenshot of a git bash command window running on your laptop.

The otherwise unsupported assignment was consistently completed independently and in under 10 minutes.

4.2 Personal Repositories

To introduce students to personal software repositories, students created a repository and pushed their final assignment submission code to their new repository.

1. Complete the assigned project.
2. Create a new private repository with the same name.
3. Commit and push your code up to your private repository.

Again, the additional time required was very little—and aside from simple directions, little lecture or other classroom time was required. At this time, only Git was required (most students used Git for Windows). TortoiseGit was recommended, but students were encouraged to explore (and report on) their experiences with other tools. Students reported positive experiences with SourceTree, Git Bash (command line), and the integrated Git in Microsoft's Visual Studio.

Git was incorporated into several other personal assignments and the students were assigned a discussion forum to comment on and share their experiences. Students shared resources and worked together to learn, teach, educate, and persuade others to explore various approaches.

4.3 Full Collaboration

Version control is useful for personal projects, but additional benefits occur in collaborative projects. After several personal assignments and shared discussion forums, we were ready to introduce collaboration. At this point, we would have 36 students adding code to a shared application. As experienced developers may know, adding files to a common code base introduces a variety of common situations including: Added, but missing code files, code conflicts, disappearing edits, and complicated merges.

When code files are completely separate from others, problems are minimal. However, when multiple users edit the same code file—especially on or near the same lines of code, the problems become more challenging.

We introduced both situations by having the students add a top-level menu item for each team. Each team menu item had a sub-menu item for each student taking the user to their own profile page with custom images, css, styles, and appearances. Student-specific pages of course, created minimal conflicts (except sometimes for over-zealous styling), and the shared menu pages created a small, but targeted location to address the more complex issues associated with mutually-edited lively-evolving entities, or *melees*.

This assignment focused on *Creating Views* (and working collaboratively with a large project team).

1. Clone the code from the reference project.
2. Right-click on your projects folder and select *Git Bash Here*. Type the provided URL and hit Enter.
3. In Windows Explorer, right-click on the new solution file to open the project in Visual Studio.
4. When you run the program, more information about this project will show up on the opening page.
5. Add one top-level menu item for each team. Each developer will have their own menu item under their team (with their own menu item, controller action, and view).
6. Add yourself to the menu so users can access your page.
7. Add an action method to the associated controller.
8. Add a view to be returned as your action response.
9. Display any (school-appropriate) content you like on your personal page.
10. See examples under the *Teams* menu option to get started.
11. First, make a small change and then commit your code back into the master branch and push it to the repo. Start with a change to the README.md if you want to be safe.
12. Remember to **add** any new files (e.g., if you included images) before committing your code to your local repository and pushing to the shared master branch.
13. Don't break the build; code can be committed in a partial state but it must compile when you commit.
14. Commit and push small changes. Everyone will be adding their name to the next slot and you could have terrible problems with merging if you don't (a) pull the newest version (b) commit and push your menu changes quickly.
15. Pull new code frequently; the project will be changing quickly as many team members work on a short deadline.

16. Your goal should be to have at least your name added to the menu by the end of our class period tomorrow.

Additional experiences were included throughout the course, but this assignment provided a realistic, or perhaps, even more challenging experience than the typical introduction to Git.

Results from these series of small steps to introduce Git in the native environment for students new to version control are provided in the results section.

5 Evaluating the Introduction of Version Control

Student response to both approaches was measured with a survey. Response to the incremental native introduction of git was given to and completed by 36 students in a graduate-level applied computer science web applications course using ASP.NET 5 (now ASP.NET Core 1.0), MVC 6, and Entity Framework 7. Students varied in programming ability and exposure to version control. Response to the lessons using GitSubmit was given to a smaller class of undergraduate students in a second-year course for new programmers working on their Bachelors of Computer Science. Students were primarily novices in both programming and version control. Five of eleven students completed a preliminary survey; three of these five completed a later version as well.

Students learning Git in a native environment were generally older; all were over 21 and 31% were over 25. 16 of 36 were in their first year of graduate school and 16 were in their second year. 11% were in their third year. Nine participants (25%) were female. Twenty (55%) are first-generation college students.

Students learning git with GitSubmit were generally younger; all were under 25 and three of the five were under 21. Two were juniors and three were seniors. All were male. One was a first-generation college student.

6 Results

Responses to both approaches were positive. On a 6-point Likert scale, students most commonly agreed or strongly agreed with statements such as *the process was easy to use*. For scoring, the following values were used [1].

No answer	1
Strongly disagree	2
Disagree	3
Neutral	4
Agree	5
Strongly agree	6

[1]Given prior experience, we did not anticipate the extent to which the responses would be positive. In later surveys, we may expand the scale to allow more discernment between positive responses.

Student response to the seven general reaction statements below are shown in Figure 6.

1	The process will be applicable to my intended profession.
2	I would recommend this process.
3	I would like to use this process on future assignments.
4	I view this process in a positive light.
5	The process was convenient and an efficient use of time.
6	The process was easy to use.
7	The process distracted from core content and my intended learning.

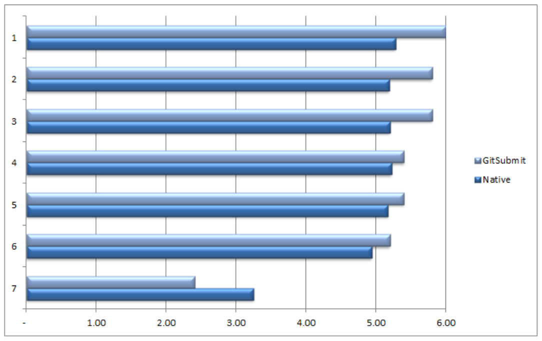


Figure 6: Student response to version control introduction.

Students using GitSubmit to submit their assignments preferred it to the current submission process in our existing LMS and recommended that we propose the process to classes outside computer science. Open-ended comments included:

- *Much more reliable and easier to use compared to [our current LMS].*
- *Keeps track of versions of your work and has the ability to update your work (for revisions and such).*
- *The layout is very logical, especially if you have used version control software before.*
- *It was quick to get all assignments at once.*
- *Easy to grab since it is on my desktop.*
- *I think this program is a fantastic idea, a great solution to the clumsy [current LMS], and I'm sure that it would be useful in many other departments.*
- *It's better overall than [our current LMS].*

Students working with the incremental native lessons provided many comments about things they liked; 35 of 36 students provided open-ended feedback:

- *It was very helpful in collaborating on a project.*
- *Collaboration work can be done easily* [2].
- *Can work on the section of the project individually and collate the work with the team.*
- *Merging and stashing.*
- *Code merging between team members working on different modules.*
- *Easy to commit the code and use it by all.*
- *Easy commands to clone and pull.*
- *You don't have to keep track of all the changes your teammate made in his version.*
- *Easy to clone the project by using GitBash Here.*
- *Branching and merging are easy.*
- *Workflow is flexible.*
- *Good process for learning how to keep a track of your work in the project.*
- *Maintaining the code at one place.*
- *Git was quite easy and it would show all the commands to do for pushing and pulling the code.*
- *Share the code between team members.*
- *Process was easy for initial setup as well as in long run.*
- *It is easy when working on a project. Everyone can work together on the code.*
- *Helps greatly in collaborating to continue the workflow.*
- *Easy to share code and keep teammates updated*
- *Getting acquainted with command line interface.*
- *It is easy to share code and work on it; [whenever] we need we can easily work on code.*
- *It helps in sharing knowledge and also to complete projects.*
- *Fast and progress can be seen.*
- *Easy to use.*
- *I have never tried other tools for the same purpose but the Git looks pretty much easy and efficient.*

28 students offered a comment when asked what they disliked. Sixteen of those students mentioned merge conflicts as their only dislike. Others mentioned initial difficulties, *It was a bit difficult to learn in the start but it made collaboration much easier as I got to know it* and a couple each mentioned difficulties learning commands and synchronizing with the servers.

[2]Ease of collaboration mentioned in 8 responses

7 Conclusions

This paper demonstrates the introduction of version control to novice students in two ways, first through a scaffolded process for those new to both coding and version control and secondly through an incremental process in the native environment for those with some coding experience. The most troubles for native users came from learning commands, understanding the synchronization with the servers, and dealing with merge conflicts, all areas in which GitSubmit provides scaffolding support. Results were positive with students finding version control both very applicable to their profession and, perhaps more surprising for first experiences with Git, easy, useful, and enjoyable to use.

From the initial tests, it appears that GitSubmit may push positive scores to an even higher level. We will continue to develop and extend the integration of Git—via GitSubmit and later via the native environments—into our computer science curriculum.

Further work includes integration and testing of GitSubmit in additional introductory computer science programs at two universities. Concurrently, additional definitions are being added to the associated visual language and the development of additional tools for instructor support for managing and grading large numbers of coding assignments will continue. The progressive lessons will be extended to incorporate additional alternatives such as SourceTree and GitHub Desktop, additional work with native environments including the Git Bash command line, and the further development and testing of GitSubmit. Development of a reusable library of progressive version control assignments will continue, expanding the library to provide set of short, flexible lessons that can be integrated across a variety of existing courses.

References

[1] N. W. Eloe, D. M. Case, and J. L. Leopold, "VeCVL: A Visual Language for Version Control," in *Proceedings of the 2016 International Workshop on Visual Languages and Computing (in conjunction with the 22nd International Conference on Distributed Multimedia Systems (DMS'16))*, 2016.

[2] M. Guzdial, "Software-realized Scaffolding to Facilitate Programming for Science Learning," *Interactive Learning Environments*, vol. 4, no. 1, pp. 001–044, 1994.

[3] D. Wood, J. S. Bruner, and G. Ross, "The Role of Tutoring in Problem Solving," *Journal of Child Psychology and Psychiatry*, vol. 17, no. 2, pp. 89–100, 1976.

[4] S.-K. Chang, "Preface to volume 1," in *Journal of Visual Languages and Sentient Systems*, vol. 1, 2015, p. 4.

[5] S. K. Chang, *Visual languages*. Springer Science & Business Media, 2012.

[6] M. Coccoli, A. Guercio, P. Maresca, and L. Stanganelli, "Smarter universities: A vision for the fast changing digital era," *Journal of Visual Languages & Computing*, vol. 25, no. 6, pp. 1003–1011, 2014.

[7] N. M. Consortium *et al.*, "Nmc horizon report 2014 higher education edition," 2014.

[8] M. Coccoli, P. Maresca, and L. Stanganelli, "Enforcing team cooperation: an example of computer supported collaborative learning in software engineering," in *Proceedings of the Sixteenth International Conference on Distributed Multimedia Systems*, 2010, pp. 189–192.

[9] D. Linville, M. P. Rogers, C. Kelly, C. Spradling, and D. Case, "Profession-based learning through collaboration and vertical alignment with k12, higher education, and industry: panel discussion," *Journal of Computing Sciences in Colleges*, vol. 31, no. 5, pp. 187–189, 2016.

[10] M. Coccoli, L. Stanganelli, and P. Maresca, "Computer supported collaborative learning in software engineering," in *2011 IEEE Global Engineering Education Conference (EDUCON)*. IEEE, 2011, pp. 990–995.

[11] A. Meneely and L. Williams, "On preparing students for distributed software development with a synchronous, collaborative development platform," in *ACM SIGCSE Bulletin*, vol. 41. ACM, 2009, pp. 529–533.

[12] R. Francese, C. Gravino, M. Risi, G. Scanniello, and G. Tortora, "On the Experience of Using Git-hub in the Context of an Academic Course for the Development of Apps for Smart Devices," in *Proceedings of the 21st International Conference on Distributed Multimedia Systems (DMS'15)*, 2015, pp. 292–299.

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

[14] 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.

[15] V. Isomöttönen and M. Cochez, "Challenges and confusions in learning version control with git," in *Information and Communication Technologies in Education, Research, and Industrial Applications*. Springer, 2014, pp. 178–193.

[16] GitLab, "Code, test, and deploy together with GitLab open source git repo management software," https://about.gitlab.com.

[17] S. Chacon, *Pro Git*, 2nd ed. Berkely, CA, USA: Apress, 2014.

[18] Atlassian, "Bitbucket Cloud Documentation Home," https://confluence.atlassian.com/bitbucket/bitbucket-cloud-documentation-home-221448814.html.