ABSTRACT
Numerous CS faculty have demonstrated the benefits of using version control in courses other than software engineering. However, they maintained their own servers, and to the best of our knowledge, none published experiences with version control in CS1 courses for non-CS engineering majors. As a result, even faculty experienced with version control may hesitate to adopt it in some courses, fearing that it is too difficult, time consuming or distracting. In this paper, we describe how we adopted version control in a CS1 course for non-CS engineering majors, and how project hosting services facilitated its use. Our experience indicates that undergraduate engineering majors in CS courses can gain competence in version control, and project hosting services simplify course management.

Categories and Subject Descriptors
K.3.1 [Computer Uses in Education]: Computer managed instruction; K.3.2 [Computer and Information Science Education]: Computer science education

General Terms
Education, Management

Keywords
Version control; course management; software engineering

1. INTRODUCTION
Version control is essential to software development [11, 15, 21] and beneficial in the classroom [6, 10, 12, 18], yet few students gain sufficient experience through undergraduate courses alone [18, 24]. One industry source stated: “I’ve worked with some twenty-odd interns […] all of them graduate students at top electrical engineering and computer science departments from around the world. I’ve also interviewed many more job candidates. One fact that has continually amazed me is what a tiny fraction of them have had any previous experience using version control systems. Not only will using version control help you write better software, but it will help you get a job if you ever want to go into industry.” [24]. One student replied: “As an undergraduate I can tell you from experience that there are virtually no classes that expect you to use [version control]” [24]. In an internal survey of Wentworth computer science students in 2012, one said that we need to emphasize in our curriculum: “More mobile development. More micro-controller development. VERSION CONTROL” (emphasis theirs).

Students should gain experience in version control, but often do not for lack of faculty experience, time, or early emphasis in the curriculum. Faculty may avoid version control for lack of experience: Clifton et al. noted that only one faculty (and none of the teaching assistants) in their department had prior experience [6]. Experienced faculty may avoid version control in the classroom for lack of time: some authors noted the administrative overhead necessary to set up and manage servers [6, 18]. Faculty may also avoid version control outside of software engineering courses because the ACM Computer Science Curriculum [1, 2, 3] places coverage of the topic in a three hour section called “Software Engineering Tools and Environments.” Indeed, Reid and Wilson noted that “undergraduates in CS at the University of Toronto were not introduced to version control until their final-year software engineering course. As a result, most students completed their degree with only a hazy understanding of version control’s importance and how best to use it” [18].

In this paper, we do not focus on the benefits of version control in the classroom (as others have already done); instead, we examine how to overcome the obstacles that prevent faculty from using it more frequently. For the inexperienced, this paper briefly reviews relevant background knowledge, surveys related work, cites tutorials, and describes how to set up and use version control in the classroom, step-by-step. For the time-strapped, this paper explains how cloud project hosting services reduce administrative overhead. For faculty who avoid version control outside of software engineering courses, this paper describes how non-CS engineering freshmen successfully used it for work submission in CS1. This paper also describes our experience with version control in object oriented programming, compiler design, and software engineering courses. Finally, this paper calls for the ACM Computer Science curriculum to properly reflect the importance of version control by emphasizing early and repeated exposure to it.
2. BACKGROUND AND RELATED WORK

2.1 Version control

Version control systems enable multiple contributors to record changes (make commits) to a project over time. Obsolete version control systems locked files to ensure that only one contributor could edit at a time. Modern version control systems (e.g., CVS, Subversion, Git, Mercurial) handle concurrent contributions by merging changes together. Merging not only facilitates concurrent contributions, but also enables organizing commits by topic (branch) or contributor (fork). If changes overlap (conflict), then the contributor must resolve the conflict manually. Free training resources are available for all of these systems.\(^1,2\)

Version control systems are either centralized or distributed. Centralized version control systems (such as CVS and Subversion) store the project history (repository) on a remote server, not locally. In centralized systems, making a commit record changes to the server; consequently, contributors must merge any recent changes from the remote server (and resolve any conflicts) before making a commit. Distributed version control systems (such as Git and Mercurial) include all of the operations of centralized systems, but decouple recording changes from publishing them by storing project history locally and on remote servers. In these systems, making a commit records changes locally; contributors must then push to publish commits to a server and fetch or pull to acquire commits from others. Users of distributed version control systems can always make commits off-line without conflicts and decide when (and if) to merge later.

Distributed version control systems are more robust and secure than centralized systems. While centralized systems number commits sequentially, distributed systems use secure hashing algorithms to identify commits and detect tampering with history, such as students rewriting logs to “give the impression that they had completed work earlier than they actually did”, as is possible with CVS [18]. Distributed systems store data in a hidden folder in the project root, while centralized systems store data in every subdirectory of the project (e.g., CVS/ or .svn/), making it easy for students to confuse these folders with the working directory and damage both [18]. In the event of a server failure, users of centralized systems lose project history and the ability to make commits; users of distributed systems lose nothing.

2.2 Version control in the classroom

Faculty have noted the benefits of version control for assigning, collecting, and returning student work [6, 12, 18]. Posting assignments using version control enables students to start with skeleton code and test cases, and submit work repeatedly to faculty at any time before or after the deadline, since submissions have timestamps. Version control makes it easy to post grades, give feedback on work in progress [6, 12], review changes (diffs), work remotely across multiple machines [16, 18] or assist during office hours [18].

Using version control offers other pedagogical benefits. Version control makes it easy to identify problems in student work habits [6, 12, 18], collaborate in group projects [10, 12, 18], assess individual contributions in group settings [6], coordinate among faculty and teaching assistants [6], and back up work off-site [12]. Students exposed to version control early on improve work habits in time for working on team projects [18] and also build skills for their careers.

The alternatives to version control for course management are worse. Assigning or collecting work via email clutters inboxes, and students may forget (or claim spam filters ate) attachments, requiring follow-up. Collecting work on a server requires system administration. Course management systems such as Blackboard or Moodle solve these problems, but students will not use these systems outside of school.

2.3 Project hosting services

Previously, faculty who published their experiences with version control (ourselves included) administered their own servers [6, 12, 18]. Faculty administering a server can retain full control, but bear the burden of acquiring access to a server, maintaining student rosters, initializing repositories for each student, ensuring sensible permissions, giving each student secure access, and configuring a web interface so students can confirm that they pushed commits properly. Thanks to project hosting services, we no longer maintain our own servers for version control.

Cloud-based project hosting services host version control repositories on-line so that contributors on a team can share changes with each other without administering their own servers. Examples include: Assembla, Bitbucket, Github, Google Code, ProjectLocker, Sourceforge, and Unfuddle. Many services host open source projects free of charge. In addition, these services may also include an issue tracker and a wiki, similar to systems such as IBM Rational Team Concert [16] or DrProject [19], and include code review features that facilitate rapid feedback to students.

Faculty should adopt project hosting services carefully because the U.S. Family Educational Rights and Privacy Act (FERPA) requires maintaining the privacy of student educational records. Thus, it is necessary to “prevent students from seeing one another’s repositories” [18]. However, according to the U.S. Department of Education, “FERPA does not prohibit the use of cloud computing solutions for the purpose of hosting education records; rather, FERPA requires States to use reasonable methods to ensure the security of their information technology solutions” [23]. Fortunately, Assembla,\(^3\) Bitbucket,\(^4\) and Github\(^5\) offer private repositories for educational use free of charge, so that students and faculty can share only with each other, and nobody else.

3. WHY GIT ON THE CLOUD?

We chose Git due to the technical merits of distributed systems and because more project hosting services support Git directly or indirectly than any other version control system. Assembla, Bitbucket and Github all support Git; if any service fails, the class can switch to a competitor. Ensuring such continuity with Subversion would require local backups; fortunately, Git makes local backups by design and works well with Subversion repositories. Git is also the fastest version control system, and has gained market share at the expense of CVS, Mercurial, and Subversion (see Figure 1). Apple, Google, Microsoft, IBM, Facebook, Linux, Perl, Eclipse, Android, and Ruby on Rails all use Git.

---

\(^1\)http://www.ericsson.com/vcbe/html/  
\(^2\)http://software-carpentry.org/4_0/vc/  
\(^3\)https://assembla.com/  
\(^4\)https://bitbucket.org/  
\(^5\)https://github.com/
4. SETTING UP GIT ON THE CLOUD

This section shows how to set up Git on the cloud so that faculty can modify any repository in the classroom, but students cannot see or modify each others' repositories. Section 5 shows an example of using Git for course management. We do not assume familiarity with Git, but cannot cover everything: read these books [5, 13, 22] or tutorials6,7,8,9 for more information. Graphical interfaces to Git use the same names for commands as Git itself; either way, do not type in $ (the shell prompt) or instruction examples literally.

Faculty must first decide on a project hosting service. Assembla offers free private repositories regardless of purpose, but excludes code review or issue tracking. Bitbucket offers unlimited free private repositories automatically for .edu domains. Github offers free private repositories by request.10

4.1 Preliminaries for faculty and students
1. Download and install Git11 and merge tools (e.g., Meld, TortoiseGit, or KDiff3) to handle conflicts easily.
2. Set up the user name and email for Git. For example,

   $ git config --global user.name "Jane Smith"
   $ git config --global user.email jane@wit.edu

3. (Optional) Set up the commit editor. For example, to set up Notepad++ (if it is on the PATH),

   $ git config --global core.editor "'notepad++' -multiInst -notabbar -nosession"

4. Create an account on the project hosting service with the school-issued email address. Reuse the school-issued user name if possible; otherwise, collect user names from students (e.g., through Google Forms).
5. (Recommended) Create an SSH authentication key to avoid password re-entry for push and pull operations.

   $ ssh-keygen -t rsa

   Share the public key (*.ssh/id_rsa.pub) with the project hosting service and use SSH remote repository URLs; otherwise, use HTTPS URLs.

4.2 Course repository setup for faculty
1. Create the course repository. For example,

   $ git init CS1
   $ cd CS1

2. Put course materials into the new folder (e.g., in CS1).
3. (Recommended) List paths or file globs for Git to ignore in a file called .gitignore. For example,

   
   
   
   
   

4. Add the current folder to Git, and then commit.

   $ git add .
   $ git commit -m "Added course materials."

5. On the project hosting service, create a new public course repository for students to clone (e.g., CS1).
6. Add the remote repository URL to Git. For example,

   $ git remote add origin \
   git@bitbucket.org:lawrancej/cs1.git

7. Publish commits to the project hosting service:

   $ git push -u origin master

8. Instruct students to follow steps in the next subsection.
9. Add each student’s private fork to Git. For example,

   $ git remote add alice \
   git@bitbucket.org:alice/cs1.git
   $ git remote add bob \
   git@bitbucket.org:bob/cs1.git

10. (Recommended) Verify the class set up Git properly.

   $ git remote -v # Is the roster complete?
   $ git fetch --all # Did students push?

4.3 Setup for students
1. On the hosting service, create a new private repository (e.g., CS1). Do not fork the course repository.
2. Then, add faculty as the sole collaborator so faculty can view work, offer feedback and post grades.
3. Download (clone) the public course repository, and enter the folder. For example,

   $ git clone \
   https://bitbucket.org/lawrancej/cs1.git
   $ cd cs1

4. Add your remote URL and push to it. For example,

   $ git remote add my \
   https://alice@bitbucket.org/alice/cs1.git
   $ git push -u my master

**If available, give administrator privileges to faculty.**
5. USING GIT: AN EXAMPLE

Figure 2 depicts the steps; Table 1 lists useful commands.

1. (Faculty) Commit to a new branch (e.g., `assignment1`), to organize submissions and prevent merge conflicts.

   ```
   $ git checkout -b assignment1 master
   $ echo "By Monday, write a C program to print: Hello, world." > assignment1.txt
   $ git add assignment1.txt
   $ git commit -m "Assignment 1"
   ```

2. (Faculty) Push to the course repository (`origin`).

   ```
   $ git push origin assignment1
   ```

3. (Students) Fetch and check out `assignment1` from `origin`.

   ```
   $ git fetch origin
   $ git checkout -b assignment1 origin/assignment1
   ```

4. (Students) Complete and commit the assignment.

   ```
   $ echo "#include <stdio.h>
   int main (int argc, char ** argv) {
       printf("Hello, world.\n");
   }
   " > assignment1.c
   $ git add assignment1.c
   $ git commit -m "Assignment 1 solution"
   ```

5. (Students) Push work to your private repository.

   ```
   $ git push my assignment1
   ```

6. (Faculty) Collect (fetch) submissions at the deadline.

   ```
   $ git fetch --all
   ```

7. (Faculty) Checkout a student's submission (e.g., alice).

   ```
   $ git branch -D assignment1
   $ git checkout -b assignment1 alice/assignment1
   $$ less *.c
   ```

8. (Faculty) Commit feedback or grades.

   ```
   $ echo "Good job. Grade: A" >> assignment1.txt
   $ git commit -a -m "Feedback for Alice."
   ```

9. (Faculty) Push `assignment1` feedback to the student. Repeat steps 7-9 for each student.

   ```
   $ git push alice assignment1
   ```

10. (Students) Pull feedback from faculty.

    ```
    $ git checkout assignment1 # Switch to branch
    $ git pull my assignment1 # Fetch and merge
    $ less assignment1.txt # Review feedback
    ```

Table 1: Useful Git commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>git status</code></td>
<td>Show status</td>
</tr>
<tr>
<td><code>git diff</code></td>
<td>Show unstaged changes</td>
</tr>
<tr>
<td><code>gitk --all &amp;</code></td>
<td>Show commit logs</td>
</tr>
<tr>
<td><code>git gui &amp;</code></td>
<td>Git graphical interface</td>
</tr>
<tr>
<td><code>git merge branch</code></td>
<td>Merge in branch</td>
</tr>
<tr>
<td><code>git mergetool</code></td>
<td>Handle conflicts</td>
</tr>
<tr>
<td><code>git checkout --theirs file</code></td>
<td>(Merging) Use their file</td>
</tr>
<tr>
<td><code>git reset --hard HEAD</code></td>
<td>Set aside (stash) changes</td>
</tr>
<tr>
<td><code>git shortlog -s</code></td>
<td>Count commits by author</td>
</tr>
<tr>
<td><code>git push -f remote</code></td>
<td>Force a push to remote</td>
</tr>
</tbody>
</table>
6. CLASSROOM EXPERIENCES

We learned best practices for using version control systems with project hosting services from experience with software engineering, compiler design, and object-oriented programming courses before using it in CS1 for engineering majors.

6.1 For computer science majors

Although faculty can use version control on the cloud for course management (and that is a good way to get started), version control lends itself to complex project-based pedagogy, since it tracks work for real-world projects. In short, students learned well by collaborating with each other.

In our software engineering course, we asked student teams to select and improve an existing open source project. Because Git can interface well with almost every other version control system, students were able to choose any project. Version control was essential to run this course: students needed it to collaborate, and we needed it to distinguish student work from upstream project contributions.

In our compiler design course, we asked student teams to write a toy compiler and also contribute features to a class-wide project. From this experience, everyone (ourselves included) learned how good habits such as topic branching and small, incremental commits reduced the risk of merge conflicts. Many students even contributed fixes to the course materials themselves and asked us to pull in their changes.

We discovered that with distributed version control, because we did not need to merge in changes, it facilitated conversations about whether code was good enough to be merged in. One student in this course told us: “learning Git was probably the most useful thing for me, personally.” Another student told us that they learned more about version control in this course than in their software engineering course.

In our sophomore object-oriented programming course, we asked students to review each other’s code and also collaborate in teams on projects of their choosing. Students reviewed each other’s code anonymously, learning through practice how to assess quality. Just as peer instruction is distributed, because we did not need to merge in changes, it facilitated conversations about whether code was good enough to be merged in. One student in this course told us: “learning Git was probably the most useful thing for me, personally.” Another student told us that they learned more about version control in this course than in their software engineering course.

In our compiler design course, we asked student teams to write a toy compiler and also contribute features to a class-wide project. From this experience, everyone (ourselves included) learned how good habits such as topic branching and small, incremental commits reduced the risk of merge conflicts. Many students even contributed fixes to the course materials themselves and asked us to pull in their changes.

We discovered that with distributed version control, because we did not need to merge in changes, it facilitated conversations about whether code was good enough to be merged in. One student in this course told us: “learning Git was probably the most useful thing for me, personally.” Another student told us that they learned more about version control in this course than in their software engineering course.

In our sophomore object-oriented programming course, we asked students to review each other’s code and also collaborate in teams on projects of their choosing. Students reviewed each other’s code anonymously, learning through practice how to assess quality. Just as peer instruction is more effective than traditional lecture [14], we found that code review engaged peers to learn from each other. In an unsolicited email, one student in this course said:

“I’ve been meaning to thank you again for being such a progressive teacher as when I was in your class having us use Git, and talking to us about unit testing. My understanding of Git was very important in me getting an awesome summer dev job. Thank you! I even use Git to host my own code so I can code on multiple computers without an issue. I just want to say keep up with the way you’re teaching, I wish other teachers were as progressive about using technology like Git. I’m sure other teachers would benefit from using Git as a submission system, or even have the students post onto a personal or school Github.

Another student in this course told us:

I just got back from an interview, and the guy interviewing me asked me if I knew Git. I said yes, in fact, we used it in the Java course. The interviewer said: “Thank your professor.”

6.2 For engineering majors in CS1

In our search through the SIGCSE literature, we found no published experiences with using version control in CS1 courses for non-CS engineering majors.

In our CS1 course, we asked electrical and mechanical engineering freshmen to complete individual assignments and in-class activities in C, using version control for course management as described in Section 5. As in our courses for computer science majors, we adopted the Gutenberg method of teaching by answering questions about the reading in class [17] and the Thayer method of teaching by moving work into the classroom [20]. We programmed in front of the class, and students often typed what we typed. When students asked for our code, we committed to a separate branch to avoid conflicts with their work, and asked them to commit or stash their work first before pulling from us.

In our experience, almost all students were able to learn enough Git commands to receive starter code and submit the first lab on time; however, students often needed reminders about how to use version control. Prior to every deadline, we reminded students to commit first before pushing. When students submitted work before and after the deadline, we asked them to tag the work to be graded; that way, they could decide whether to suffer any late penalty.

7. LESSONS LEARNT AND DISCUSSION

Although distributed version control often confused students, they had little difficulty in adopting its workflows and never asked “when am I ever going to use this?” Students often made multiple commits off-line between pushes, and students on teams who pushed to a single remote repository (adopting a centralized workflow) encountered more difficulty than those who pushed to their private forks and merged later (adopting a distributed workflow). Students appreciated using Git, and did not feel learning it was wasted time. In fact, several of our students continued to use Git outside of our courses.

Despite the benefits of version control systems and project hosting services, they are unsuitable for some uses. Version control systems work well for human-readable text files, but are cumbersome for file formats such as Word, Excel or PowerPoint; cloud services such as Google Drive support collaboration better for those formats. Version control systems do not notify collaborators of changes in real-time (except for Jazz [16]), and cannot replace conversations about revisions. Project hosting services cannot fully replace systems such as Blackboard or Moodle, since they lack features such as online quizzes or mailing lists for announcements; however, cloud services such as Google Forms and Google Groups can serve these purposes, respectively.

We believe that the ACM Computer Science Curriculum should move version control from software engineering into programming fundamentals, to reflect the changing face of computing. Currently, faculty may hesitate to teach students version control outside of software engineering, thinking that it is either too difficult, time consuming, or would distract attention away from relevant course topics. When we tutored sophomores who struggled to collaborate on a project because they were unaware of version control, we decided to emphasize it in all of our programming courses. Immersing students in practice with version control in the programming fundamentals sequence benefits everyone.
8. CONCLUSION

Faculty should use version control in many science and engineering courses for students to gain experience with it. Unlike prior work that focused on computer science majors, we have demonstrated that freshmen in engineering majors can become skilled in version control as part of work submission. This success leads us to believe that science or engineering students can learn how to use version control. Furthermore, unlike prior work in which faculty maintained their own servers, students used private repositories on a cloud project hosting service to coordinate and submit work. We found that using a project hosting service was no more difficult than using existing course management systems. Therefore, we believe that faculty have no reason to avoid using Git on the cloud in the classroom.

9. ACKNOWLEDGMENTS

We thank our anonymous reviewers for their useful feedback. In particular, reviewer B’s numerous suggestions and criticisms helped us improve this paper.

10. REFERENCES
