

Challenges and Confusions in Learning Version Control with Git

Ville Isomöttönen and Michael Cochez

Department of Mathematical Information Technology
University of Jyväskylä
P.O. Box 35 (Agora), 40014, Jyväskylä, Finland
{ville.isomottonen, michael.cochez}@jyu.fi

Abstract. Scholars agree on the importance of incorporating use of version control systems (VCSs) into computing curricula, so as to be able to prepare students for today's distributed and collaborative work places. One of the present-day distributed version control systems (DVCSs) is *Git*, the system we have used on several courses. In this paper, we report on the challenges for learning and using the system based on a survey data collected from a project-based course and our own teaching experiences during several different kinds of computing courses. The results of this analysis are discussed and recommendations are made.

Keywords. Version control systems, Git, Computer Science Education

Introduction

Version control systems (VCSs) have been actively used since the '70s. The older tools like Source Code Control System (SCCS) [22] and Revision Control System (RCS) [23] only supported storage of the versions on a locally mounted file system. Later systems were characterized as centralized systems, the well-known representatives being the Concurrent Versions System (CVS) [2] and Subversion (SVN) [6], the latter of which is still extensively used. In the present day, many users have adopted distributed version control systems (DVCS) (see [7]) where each user has a local copy of the repository that can be synchronized with other repositories, introducing more flexibility, and perhaps more complexity, to the system. The decentralized technology allow for change tracking, reversibility, and manageable collaborative work. In this paper, we are concerned with learning and using DVCSs in computing education context. The system we have used is Git¹.

Previous research suggests that there is an agreement that learning and using VCSs would be important for computer science students (see, e.g., [16]). Version control systems have been used in various computing courses, while there is still a call for more empirical research on the student usage of the systems (see, e.g., [19]). We have previously studied students' use of Git by analyzing

¹ <http://git-scm.com/>

repositories with various metrics [5, 4]. In the present paper, we continue this research by focusing on student and teacher (authors) experiences. The former is based on a survey targeted to students of a project-based course, while our own experiences originate from several computing courses. Our main research perspective are the challenges the students report and we have identified through teaching.

1 On Version Control Systems in Education

Version control systems (VCS) have been adopted in a variety of educational settings. When this kind of system is used in a course, teachers try to use the tool to support their traditional teaching practices. Liu et al. [15], for instance, used the system to monitor and visualize the contributions of the members of student teams. Clifton et al. [3] used a VCS for course management including materials, assignments, and learning processes. Others, like Hartness [10], adopted version control to promote realistic software development experiences. Further, Robles and González-Barahona [20] used the version control system to detect plagiarism by students, which was made possible by the fact that the system tracks versions for all students. Besides the automated plagiarism detection, they adopted black-box testing to automatically assess the functional requirements the software has to meet every time students submitted code into the repository. Lastly, they used the code submitted to the version control system to automatically generate personalized exams. More specific to programming education, we find a work by Glassy [9]. In his work, he tries to use the VCS not only as a tool to monitor students' progress, but also to understand *how* students develop code. In general, project-based courses are perhaps the most natural habitat for VCSs that naturally support team work and can help a teacher estimate student activity for assessment and grading [25]. We also notice that use of VCSs has been required of non-CS majors [13], and that these systems are employed outside the computer education domain. For instance, Lee et al. [14] used it to manage creative writing work.

VCSs are not specifically intended for use in educational settings, and several difficulties and confusions may emerge. As noted in earlier work by Reid and Wilson [19], who used the CVS system, there might be confusion in judging which of the students' assignment versions was the final one. This can be addressed if the students adopt the tagging feature of the system [13]. Other problems that Reid and Wilson noticed was that students mix up different functionalities of the CVS, namely the check out and update commands. Also, a well developed background knowledge for the teaching assistants was deemed to be necessary. Most issues were considered to be due to the lack of a proper mental model of the VCS system used. Time and resources needed for setting up and managing repositories are yet another challenge raised by the same authors. More recently, Lawrence et al. [13] addressed this management overhead issue by using cloud services. Glassy [9] found out that VCSs do not urge students to adapt to a more iterative work progress. Instead, students tended to postpone working on

the assignments. We have earlier concluded that if a VCS is given to students as a submission tool, they might not adopt the professional use of the system but only use it for submission purposes. In general, student difficulties are reflected in the study by Robles and González-Barahona [20] who noted that when students committed code to Git, they had to ask them to upload the code to a website “just in case they did not use git in a proper way.”

Perhaps the main reason for incorporating VCSs into computing curricula, as agreed by many, is that current globalized workflows require tools which support distributed collaboration [16]. Despite the challenges summarized above, there are clear indications that students are able to adopt VCSs. With support in place, even non-CS majors have successfully used the basic functions of the system [13]. In this connection, it has been observed that Git is not a tool with a flat learning curve; Xu [25] points out that it might take a long time for students to familiarize with a VCS. Familiarity seems also to be the main reason why Rocco and Lloyd [21] found that commit frequency is increasing during a course. The authors designed two assignments. For the first one, they set a minimum commit frequency while the second one had no requirements. Still, only 75% obtained a acceptable commit frequency for the first exercise compared to 81% for the latter one. The authors note that not only were the students able to grasp the basics of the VCS (Mercurial), but they tended to continue to take advantage of the tool later on. A similar observation can be found in Milentijevic et al. [17]. Their students considered a VCS useful, but only after they got sufficiently familiar with it. In accord, as we argued in our earlier research [5], there is a need to include VCSs as an integral part of a computing curriculum. Milentijevic et al. [17] take the idea further by proposing a generalized model. This model suggests the adoption of VCSs as a support tool in project-based learning scenarios.

In the present article, we aim to contribute to this body of literature by surveying student opinions about using Git through a preliminary qualitative study. We also report detailed challenges and confusions which we have observed during our daily teaching practice across several courses. In our view, this approach adds to the studies exploring usage patterns from repository data.

2 Study

This paper is concerned with challenges that students encounter when they start learning and using Git. We addressed this concern by 1) analyzing data that were collected from a Bachelor level project course (Project) with a survey and 2) by reporting our own teaching experiences in terms of what kinds of issues can confuse students when they begin to learn the system. Our teaching experiences originate from the Project course, specific VCS supervision sessions provided during an introductory second-year software engineering course (SE Course), and two master’s level practice-oriented courses where the use of Git has been encouraged as a group work tool and all the assignments have been managed using the system. The supervision sessions during SE Course have been run

during two course instances with altogether 170 passed students, providing the principal experiential source for the present paper.

The survey of the *Project* course was a voluntary research questionnaire issued to students at the end of the course (in Spring 2013). It requested the opinion of the students on the questions below:

- Q1 (open-ended): Do you think that using the VCS was useful?
- Q2 (open-ended): What difficulties did you encounter regarding version control?
- Q3 (5-point Likert scale): Do you think that your group used the VCS in an efficient manner?
- Q4 (5-point Likert scale): Evaluate how actively you used the system.
- Q5 (open-ended): Describe how you used the VCS?
- Q6 (open-ended): Did you read the commit messages of others?
- Q7 (open-ended): Did you find the messages beneficial?

The survey was answered by 21 out of 26 students, all of whom granted a research permission. The data were tabulated into a spreadsheet file, which enabled us to have all the data available in a structured manner, while extracting the themes under particular open-ended questions and making observations about the Likert scale data. Of the open-ended questions, the data from questions Q1, Q2, Q5, and Q7 were analyzed by raising the interesting points and regularities in the respective data, which is a usual step in coding qualitative data (see [18]). With these small data, numbers (quantification) are added to the presentation only if we observed some particularly dominant aspect in the data. Question Q6 yielded quite short comments which we divided into few categories and present with frequencies in a table, similar to the Likert scale data from questions Q3 and Q4. Q7 was analyzed and is presented in connection to the closely related question Q6. The first author conducted the analysis of the survey data, while the results of the analysis were challenged and discussed together with the second author for an agreement.

We argue that reporting teaching experiences as an exploration of our own observations and experiences (our second research instrument) is a beneficial approach, as DVCSs have not yet been studied extensively in an educational setting. Then, rather than basing a study on external hypotheses sourced from the literature, it should be reasonable to report observations about teaching and learning obstacles, which can inform subsequent studies. In general, there are research approaches such as self-ethnography, which characterize our role as researchers: we are authentic participants of the research setting (educational organization) returning to remembered challenges in our daily practices (see [1]).

3 Course settings

The *Project* is a 5-credit bachelor level course whose main goal is to develop students' understanding of group processes and software processes. The course was conducted under the topical theme of Open Data. Students innovated, designed,

and implemented prototypes of Open Data software products in small groups. Work rooms were provided to students during the course to support autonomous and independent work. All the groups were guided to use a VCS system for their group work. All groups selected to use Git, perhaps because many of the students had at least tried the system during their earlier courses. Majority of the groups managed their repositories in YouSource service² of the university, while some groups used Git Hub³ and Bitbucket⁴. The course is targeted to second-year students, while the participants were in different stages of their bachelor studies. It is, however, a teacher observation that most of the students had relatively little previous experience with Git, that is, experimentation with the basic use case through fetch and merge, add and commit, and push commands.

SE Course is a 3-credit lectured course for second-year students. A course assignment done in groups and an end-of-course individual exam are required for completion. The lectures focus on project management and the phases of a software life-cycle. The mandatory course assignment is the preparation of a project plan, which is done in small groups. Mandatory supervision sessions on version control were arranged at the beginning of the course in order to encourage all the students to use the distributed version control system Git for the group assignment. During 2012 autumn, there were 88 passed attendees and regarding the autumn 2013 the number is now around 90 (at the time of writing this paper; a few groups still working on the assignment).

The supervision sessions begin with a demonstration on Git usage, after which students continue to practice the topics demonstrated. We use the command line Git client for this teaching. The supervisors are available during practicing and actively guide the students. A discussion is prompted throughout the session to be able to help all the attendees to go through the basics. The sessions have provided a good opportunity to identify the aspects that need to be emphasized in teaching, as it has turned out that one and the same aspects have emerged as difficulties in these sessions.

Master's level courses referred to here are *Service oriented architectures and cloud computing for developers* (SOA&CC) and *Design of Agent-Based Systems* (Agent). The SOA&CC course acquaints students to the use of digital services and the concept of cloud computing while the Agent course aims at introducing agent systems as a novel software development paradigm.

Both courses are worth 5 credits, but students of the SOA&CC course can make an optional individual assignment which can lead to an additional 5 credits. The courses have a somewhat similar structure: during the courses students work in small groups and all of their study time is devoted to programming the exercises. Two or three weekly sessions are arranged for the group work, during which the teacher assists the students in their work. Further, mandatory contact sessions focusing on reflective program review are arranged when a part of the course is completed. Both courses emphasize students' self-direction. The aim

² <https://yousource.it.jyu.fi>

³ <https://github.com>

⁴ <https://bitbucket.org>

is to engage students without having a traditional grading system. Instead, the task of the teacher has to generate genuine interest from the students towards the topic. In [12], the authors analyze how the teaching model used in this course motivates students.

The version control system Git is suggested to students as a group work tool. The teacher does also use it to receive submissions from the students. In the SOA&CC course, Git is also used to deploy code to Platform as a Service (PaaS) providers. For the development of the agent platform in the Agent course, students use Git as a collaboration tool; both in the group and with the course teacher. At the beginning of the Agent course, each students had to perform a preliminary task with the Git version control system in a temporary repository. Concretely, there was a simulated scenario where two development branches (their own and the teacher's) had diverged from each other. The students' task was then to merge these two lines of development into one.

4 Survey results

4.1 Q1: usefulness

All respondents (100%) reported that they experienced the use of VCS (Git) as beneficial, as they saw it as a development tool for shared work and as a group work tool for sharing and communicating. From this premise, it clearly seems that an authentic project-based, group-based, assignment makes students aware of the usefulness and necessity of a VCS. Authentic group assignment makes them learn VCS more profoundly as compared to 'copying and pasting' the Git commands during some previous courses:

During this course, I finally learned to use Git little better, as I earlier just copied the commands and used them. Now I am also able to resolve upcoming problems with it [translated from Finnish by the authors].

Students also comment that they experienced using the system as useful even though they did not take advantage of its full potential. A point where the students especially pay attention to the usefulness of the system appears to be their first attempts to integrate software functionalities advanced in their group setting.

Only negative comment in the answers to this question relates to the difficulties in using the system and an experience of the lost of work in the system (Git).

4.2 Q2: difficulties

Many students reported difficulties with the system as they worked on the same parts of the software under development. That is, they experienced conflicts that they encountered as something difficult and problematic (32%). Some comments hint that conflicts were resolved by cloning a clean repository from the remote Git machine:

Table 1. Students’ view of how efficiently they used the Git system

Do you think that your group used the VCS in an efficient manner?					
scale:	1=No	2	3	4	5=Yes
f:	2	3	7	7	2
Evaluate how actively you used the system.					
scale:	1=low	2	3	4	5=high
f:	1	3	9	6	2

The version control system looks like a simple software, but it has very complicated functions that are difficult to grasp. This I have encountered at the time of conflicts, when I am committing, and usually the fastest solution is to clone the repository again... [translated by us from Finnish to English]

Perceiving conflicts as problems is likely to indicate lack of routine in resolving them; Repeated cloning is a very unfortunate solution and likely to cause frustration.

Another difficulty reported is, unsurprisingly, the lack of routine and experience. Some students reported that they deliberately used only the basic features of the system to avoid problems (e.g. conflicts), while others report that learning to use the system takes time. Overall, it seems that the students hesitated to use some features of the system under the project course where their goal was to produce working software during a fixed time period.

Finding help in a self-directed manner for more advanced features was also reported to take time and be difficult—our interpretation is that reading Git documentation may turn out to be difficult if a mental model of the system and the logic of its commands is unclear. Some students also had difficulties in learning to use the Git clients they chose to use.

Yet another interesting point in the data was that students noticed that a VCS does not replace project management. That is, they reported that, by using this system, they did not yet have sufficient knowledge of whether other members in the group were advancing their tasks.

4.3 Q3 & Q4: efficiency

Student answers to Likert scale questions Q3 and Q4 are displayed in Table 1. We find the answers to accord with the above aspects drawn from the qualitative data. That is, the students found the system to be highly beneficial and used it, while there were some hesitation and restrictions for efficient use.

4.4 Q5: usage patterns

Students reported that they mainly adopted a usage pattern where they “pulled” at the beginning of a work session and then “pushed” in the end. Some students

Table 2. Activity in reading commit messages

Did you read the commit messages of others?		
response	f	reasons
NO	6	direct communication or tools such as IRC used, no need to
YES	11	of which 3 mentioned this to be due to the tool, i.e., commits reviewed in the web page of the Git service system or in the graphical client UI
ROUGHLY	4	of which two refer to rather accidental activity enabled by the tool used

commented that they reverted their staged changes if they noticed that they did not produce working solutions during their work session or someone appeared to have worked on the same parts of the software. Again, we would see these kinds of solutions as an unfortunate usage pattern—here, the students could have relied on branching instead of reverting their work in progress. Accordingly, some answers seem to indicate that the students dared to push their changes to a remote machine only if they considered their work to be a working solution.

We also observe that there was variation in the work style of the students as some students reported that they frequently checked for changes in the remote machine on top of executing the pulls and pushes at the ends of their work session. Overall, the students would have benefited from discussion on the good routine, e.g., concerning the flexibility and value of small commits.

4.5 Q6&Q7: commit messages

Table 2 presents student activity in reading the commit messages. Little over half of the respondents (52%) reported that they read the messages, while three of these students noted that this was caused by the tool used. Almost the same number of the respondents reported that they did not read at all the messages or skimmed them through accidentally (categories NO and ROUGHLY sum up to 47%). It seems to us that as student groups were provided with work rooms during the project course, they relied quite a lot on direct face to face communication which has lessened the experienced need for reading the messages. On the other hand, as a side note, many groups struggled with communication, making us to conclude that in particular in this project work setting students would have benefited from better routine with reading the messages.

By the analysis of the answers to Q7, we can point out more reasons under the categories of previous table. With respect to students who answered NO to reading activity (see Table 2), we now find confirmation that some of these students regarded the communication going on in their project room as their commit messages. Students in this category also not that it was better to directly look at the code. Of the students who replied YES or ROUGHLY to the question

about reading activity, some refer to only small benefits due to small group size, and also preferred looking directly at the code. Some students valued the commit messages as they clarified, located, time-stamped, and hinted about the changes made by others.

One interesting aspect emerging from the data is that commit messages are considered more useful towards the end of the projects. It may seem natural for the students to write more useful messages when the software product has grown, as it is then reasonable to provide explanation for peers about the changes committed. This is an aspect that could be studied among the professional software developers and compared with educational setting: how do a professional developer attend to commit message writing when the complexity of the product is still small?

Yet another aspects in the data were that commit messages provided a snapshot describing the project state for the students and that, altogether, the messages were difficult to be trusted without deeper looks at the code.

4.6 Summary

Taken together:

- A VCS is experienced useful and necessary, but the limited resources of a project assignment may discourage students to experiment with and take advantage of the advanced features of a VCS.
- On the other hand, authentic project assignment makes students to experience benefits to the degree that they see using the system advantageous and are willing to use it.
- The basic routine in terms of requesting the changes in the remote machine at the beginning of a work session and sharing changes in the remote machine at the end of the session appears to be well absorbed, while there seems to be variation in how frequently students interact with the remote machine. In our interpretation, students need not only conceptual explanation of the inner workings of Git, but also of what a distributed VCS implies and enables in terms of usage patterns. These students would have benefited from encouragement towards and information on the benefits of frequent commits and that branching can be useful for committing the work in progress.
- In our view, the most interesting aspect emerging from the data on commit message writing, constituting a future research question on the writing activity and informativeness, is that this writing may naturally become dependent on group size, communication distance, and the current complexity of the shared product development. The finding that students considered their presence in the work room removing a need for informative commit messages is likely to explain our previous observation of nonsensical commit messages [5], while this appears to be less prevailing in a project-based course as compared to exercise-based courses [4].

5 Teaching experiences

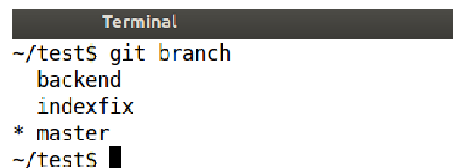
Compared to the survey results, this section provides a much more detailed view to the challenges we have encountered. Based on our own experiences as teachers, we raise several aspects that confuse or constrain student learning of Git DVCS. Notice that many of the reported challenges relate to the use of Git command line client in teaching.

5.1 Mixing Git and Bash

Student with little previous command line (unix) experience easily confuse Git commands with Bash commands, and the other way around. Especially, the commands used for file manipulation, like removing, copying, etc. cause problems. For instance, when learners are asked to remove a folder from the file system, they start writing *git*

5.2 Branches as folders or files

Manipulating branches, in particular listing them with the *git branch* command, causes learners to regard branches as folders or files in the file system (see Figure 1), and a newly initiated branch as a copy of what it was initiated from. While these conceptualizations may not be overly harmful, they seem to complicate the learners' Git usage. For instance, if asked to checkout a particular branch, learners might try to move into a branch with the *cd* command. In order to avoid this confusion, the topic should be explicitly discussed. This challenge can be in part attributed to the use of the command line client—hence, in such a setting we stress the importance of showing visual diagrams of branching and explicitly explaining branches as pointers in the commit history.

A terminal window with a dark background and light text. The title bar says "Terminal". The prompt is "~/tests\$". The command entered is "git branch". The output is "backend", "indexfix", and "* master". The prompt is now "~/tests\$ " with a cursor.

```
~/tests$ git branch
  backend
  indexfix
* master
~/tests$
```

Fig. 1. Output of *git branch* command resembles a listing of folder contents

We have also noticed learners' preference for commands like

- *git add .*
- *git add -all*
- *git commit -a*

which make their lives ‘easier’ since the repository and their directory are perfectly ‘synchronized’. However, these easy-way-out commands can turn into a habit by which student omit the questions of which of their files are reasonable to be managed with VCS and why they should also consider small and frequent commits. For instance, in our SE Course supervision sessions, we try to avoid this to occur by using more specific forms of the commands during the first demonstrations of the basic commands.

5.3 ‘git add file’ creates a new file

In Git system, changes are “staged” for the upcoming commit using `git add` command, thus there is a lot of flexibility regarding which of the current changes the user likes to commit. We have noticed a tendency where the command `git add file` is, however, associated with creating a new file, while it stages the file for the commit. We speculate that the confusion with the `git add` command simply arises from it being associated with the natural language. Considered together with the `git commit` command, which usually ensues the `add` commands, it is admittedly easy to think that “one creates a new file and then commits it”. Further, in our SE Course setting, where the command line Git client is used, we often ask learners to create arbitrary test files to practice with, which may have further encouraged this misconception. In order that learners would readily grasp the actual meaning of the `git add` command, we would include the bringing of existing files into the local Git working directory in addition to creating new ones, for instance.

5.4 origin/master versus origin master

In the Git system, the branches fetched from a remote machine are marked with `origin/branch-name`, where `origin` is an alias for a remote repository url⁵ and `branch-name` is the name of the remote branch⁶. This presentation for remote branches is used by Git when listing and merging the remote branches, for instance. However, when pushing the state of the local repository and its branch to a remote machine the ‘`origin/`’ prefix is not used, as the user gives the origin in any event in the command:

```
git push origin master:(no prefix here)master
```

A shorter and more frequently used version of this command is

```
git push origin master,
```

⁵ *origin* is a default alias for a remote repository created automatically when cloning a repository. User have the ability to assign another term as the alias or add several remotes due to the distributed nature of the system.

⁶ By default the name of the remote branch is the same as the name of the local branch, one could however choose a different name for the local branch.

where the currently checked-out local branch is pushed (imagined in front of the semicolon) to a master branch in the origin.

In referring to remote branches, this ‘with and without prefix’ conventions confuses learners until they have used the system long enough and grasped the need for specifying the remote context with a prefix in doing and showing certain tasks. In our view, this is an issue to be just repeatedly explained to learners to mitigate their potential irritableness, until the inner logics of Git commands are internatized.

5.5 No default branch on a remote machine

When creating a repository on our faculty’s Git server the repository does not contain a default master branch to start with, but a branch that the remote service needs for managing the projects and repositories internally. If the user clones the repository and starts working, the command `Git push` will not initialize a master brancher in the remote machine if not directly specified. This results in students working unintentionally with a meta-file branch in the remote machine. Here, beginners who do not yet possess knowledge of branches may encounter odd situations making Git appear a very forbidding system. Thus, the point we address here is that configurations of the Git service systems might cause great difficulties for beginners.

5.6 Accidental sub-repositories

Making sub-repositories lead to confusion and problems (most of the time by accident). For instance, students begin to experiment with `git init` command, and execute it with and without a name for the repository with the result that they end up with several repositories arranged hierarchically in their home folder tree. Students start pulling and committing but do not see changes correctly. This happens even more easily when different or several tools/terminal windows are used at the same time. In our master’s level practical courses, for instance, IDE is used for committing and terminal to check what has changed.

5.7 Blind-testing effect

The absence of a proper mental model seems to cause the blind-testing effect known in the field of the automatic assessment of programming exercises. The effect describes the situation where a student aims to make progress with arbitrary changes into a programming code, opportunistically submitting the code into the assessment system. Similarly, learners do not think about the Git commands, but “just try them out”, which makes it very difficult to interpret the responses and error messages by the Git client.

Similar to Bash commands, VIM as the default editor imposes additional cognitive load on learners, as they practice Git in the command line environment. Already the different modes of the editor cause accidental actions and

editor behavior, which occasionally irritates learners and makes them attempt to escape the editor with arbitrary key combinations. For instance, if the saving of a commit message is interrupted in VIM, an erroneous state in versioning emerges, one that requires removing an index lock in the local `.git` folder.

5.8 Absence of authentic use cases

When Git is taught as an intensive course, separate from realistic group assignments (SE Course supervision sessions), or on courses where assignments are momentary exercises (the master's level courses we refer to in this paper), there is a chance that learners associate issues such as branching with rather unorthodox use cases. For instance, they might consider that a new branch should be created for each required deliverable of the course. Moreover, typical weekly lab exercises in engineering courses communicated through systems such as a Git characterize VCS as a medium for short term storage. We have concluded earlier that using a VCS as a submission management tool, while useful, may cause learners to exactly use it as such a tool instead of a one for distributed group work [5]. In the same vein, momentary repositories created for weekly lab session run a risk of advertising systems such as a Git unfavorably. The reason is that creating a repository always involves a small amount of overhead which is not significant if the benefits of the system are clear. We also noticed in our earlier research that students who are working in a project course produce better commit messages as when they use Git for weekly course work [4] (in the present paper we analyzed the students' own view of the messages). We recommend teachers to pay attention to how to communicate course management use cases and more realistic VCS use cases to learners, and even to use another kinds of tools for pure submission returns.

5.9 Attitudinal barriers

Due to initial apparent complexity of a DVCS tool, the learners make comparison to the systems where real-time shared writing is possible in a WYSIWYG environment. We have instructed Git using the command line client, which in part may explain learners' dislikes, as graphical UI/interface is and would be a much more typical environment for many present-day learners. Further, if Git is practiced in short intensive sessions using small arbitrary test files, student are not yet shown the capacities of the system, and comparisons to cloud services such as Google Docs are very likely to occur. We have tried to overcome this barrier by discussing software engineering-oriented use cases with the learners, e.g., managing large amount of code in a distributed project. In our experiences, this discussion should occur and can remove the attitudinal barrier. In the agent based systems course (master's level), students were asked during the first week to push their answers for certain programming tasks to the repository. The teacher also created a repository which contained possible answers for the tasks. The students were asked to merge the two 'development branches' during

the second week. This task was received very well and students understood the usefulness of multiple remotes and branches.

6 Conclusions

In this paper, we have presented challenges in using Git DVCS in a project-based course based on a research survey targeted to students. Through the analysis of the primarily qualitative data, several viewpoints describing challenges experienced by the students were raised. We complemented this analysis with various detailed issues based on our teaching experiences collected from several different courses.

From the research survey, we concluded that while a realistic project course makes students to value the use of a VCS, there are also constraints such as the limited resources available during a product-oriented course, which may hamper independent learning about a version control tool. Regarding the commit messages, a future research question emerged regarding to what extent writing explanatory commit messages is dependent on group size, communication distance, and complexity of the product.

Our teaching experiences have made us consider whether viable mental models can arise from practice-first approach. To some extent, we have taught Git using theory and practice together, the approach of which is also echoed in recent pedagogy, e.g., in integrative pedagogy [24]. During SE Course supervision sessions, for instance, we have illustrated several computers on a whiteboard and connected basic Git commands to such a picture, and yet demonstrated this scheme to students with two teachers and computers. It seems to us, through the challenges reported in this paper, that the further one goes from basic features and use cases, the more important it is to communicate to student what really takes place in Git internally. For example, we have explained branches to students conceptually through the ideas such as ‘switching’ the view in the commit history. In retrospective, using this particular example, we would explicitly speak of pointers in the commit history to avoid delivering a view of branches as folders or local copies of folders. In general, the challenges we found while teaching Git could be compared to those observed when teaching other VCSs. This could reveal whether the challenges are specific to Git, its distributed nature, the command line interface, or version controlling in general.

Several small challenges we reported in the section for teaching experiences make us compare the research on learning and using DVCSs with the research on beginner programming difficulties where the mental models held by students has been studied to identify and react to misunderstandings among learners. Thus, we consider that similar methods could be used to advance the research reported here, e.g., think-aloud problem-solving [11] and phenomenography [8].

We have studied student use of DVCS (Git) through multiple courses and iterations of those courses, and, overall, are confident that students are able to adopt skills needed early, while challenges at different levels tend to remain without a constant use. Consider here, as an example, our observation of stu-

dent hesitation to use the advanced features of Git under limited resources of a product-oriented project-based course. In our opinion VCS tools need to be integrated throughout the curriculum, which implies that attention should be paid to awareness and education of department personnel and teaching assistants, so as to promote this professional skill in a continuous and gradual manner among the students.

References

1. M. Alvesson. Methodology for close up studies – struggling with closeness and closure. *Higher Education*, 46(2):167–193, 2003.
2. P. Cederqvist. Version management with CVS. Technical report, Linköping, Sweden: Signum Support AB, 1993.
3. C. Clifton, L. C. Kaczmarczyk, and M. Mrozek. Subverting the fundamentals sequence: Using version control to enhance course management. *SIGCSE Bull.*, 39(1):86–90, Mar. 2007.
4. M. Cochez, V. Isomöttönen, V. Tirronen, and J. Itkonen. *How Do Computer Science Students Use Distributed Version Control Systems?*, pages 210–228. Springer, 2013.
5. M. Cochez, V. Isomöttönen, V. Tirronen, and J. Itkonen. The use of distributed version control systems in advanced programming courses. In *Proceedings of the 9th International Conference on ICT in Education, Research and Industrial Applications: Integration, Harmonization and Knowledge Transfer*, volume 1000, pages 221–235. CEUR, 2013.
6. B. Collins-Sussman, B. W. Fitzpatrick, and C. M. Pilato. *Version Control with Subversion*. O’Reilly Media, Sebastopol, CA, 2004.
7. B. de Alwis and J. Sillito. Why are software projects moving from centralized to decentralized version control systems? In *Proceedings of the 2009 ICSE Workshop on Cooperative and Human Aspects on Software Engineering*, CHASE ’09, pages 36–39, Washington, DC, 2009. IEEE Computer Society.
8. A. Eckerdal, M. Thuné, and A. Berglund. What does it take to learn ‘programming thinking’? In *Proceedings of the first international workshop on Computing education research*, ICER ’05, pages 135–142, New York, NY, 2005. ACM.
9. L. Glassy. Using version control to observe student software development processes. *J. Comput. Sci. Coll.*, 21(3):99–106, Feb. 2006.
10. K. T. N. Hartness. Eclipse and CVS for group projects. *J. Comput. Sci. Coll.*, 21(4):217–222, Apr. 2006.
11. G. L. Herman, L. Kaczmarczyk, M. C. Loui, and C. Zilles. Proof by incomplete enumeration and other logical misconceptions. In *ICER ’08: Proceeding of the fourth international workshop on Computing education research*, pages 59–70, New York, NY, 2008. ACM.
12. V. Isomöttönen, V. Tirronen, and M. Cochez. Issues with a course that emphasizes self-direction. In *Proceedings of the 18th ACM conference on Innovation and technology in computer science education*, ITiCSE ’13, pages 111–116, New York, NY, 2013. ACM.
13. 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*, SIGCSE ’13, pages 639–644, New York, NY, 2013. ACM.

14. B. G. Lee, K. H. Chang, and N. H. Narayanan. An integrated approach to version control management in computer supported collaborative writing. In *Proceedings of the 36th annual Southeast regional conference*, ACM-SE 36, pages 34–43, New York, NY, 1998. ACM.
15. Y. Liu, E. Stroulia, K. Wong, and D. German. Using CVS historical information to understand how students develop software. In *MRS 2004: International Workshop on Mining Software Repositories*, 2004.
16. A. Meneely and L. Williams. On preparing students for distributed software development with a synchronous, collaborative development platform. In *Proceedings of the 40th ACM technical symposium on Computer science education*, SIGCSE '09, pages 529–533, New York, NY, 2009. ACM.
17. I. Milentijevic, V. Ciric, and O. Vojinovic. Version control in project-based learning. *Computers & Education*, 50(4):1331–1338, 2008.
18. M. B. Miles and A. M. Huberman. *Qualitative Data Analysis: A Sourcebook of New Methods*. Sage, Beverly Hills, CA, 1984.
19. K. L. Reid and G. V. Wilson. Learning by doing: Introducing version control as a way to manage student assignments. In *Proceedings of the 36th SIGCSE technical symposium on Computer science education*, SIGCSE '05, pages 272–276, New York, NY, 2005. ACM.
20. G. Robles and J. Gonzalez-Barahona. Mining student repositories to gain learning analytics. an experience report. In *Global Engineering Education Conference (EDUCON)*, pages 1249–1254. IEEE, March 2013.
21. D. Rocco and W. Lloyd. Distributed version control in the classroom. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, SIGCSE '11, pages 637–642, New York, NY, 2011. ACM.
22. M. Rochkind. The source code control system. *Software Engineering, IEEE Transactions on*, SE-1(4):364–370, Dec. 1975.
23. W. F. Tichy. RCS — a system for version control. *Software: Practice and Experience*, 15(7):637–654, 1985.
24. P. Tynjälä. Perspectives into learning at the workplace. *Educational Research Review*, 3:130–154, 2008.
25. Z. Xu. Using git to manage capstone software projects. In *ICCGI 2012: The Seventh International Multi-Conference on Computing in the Global Information Technology*, pages 159–164. IARIA, 2012.