The Use of Distributed Version Control Systems in Advanced Programming Courses

Michael Cochez, Ville Isomöttönen, Ville Tirronen and Jonne Itkonen

Department of Mathematical Information Technology
University of Jyväskylä
P.O. Box 35 (Agora), 40014, Jyväskylä, Finland

{michael.cochez, ville.isomottonen, ville.tirronen, jonne.itkonen}@jyu.fi

Abstract. Version Control Systems are essential tools in software development. Educational institutions offering education to future computer scientists should embed the use of such systems in their curricula in order to prepare the student for real life situations. The use of a version control system also has several potential benefits for the teacher. The teacher might, for instance, use the tool to monitor students' progress and to give feedback efficiently. This study analyzes how students used the distributed version control system Git in advanced programming related courses. We also have data from a second year course, which enables us to compare between introductory level and master's level students. We found out that students do not use the system in an optimal way; they do not commit changes often enough and regard the version control system as file storage. They also often write commitmessages which are meaningless. Further, it seems that in group work settings there is usually one dominant user of the system.

Keywords. Programming Education, Version Control System, Git

Key terms. ICTTool, TeachingProcess, ICTEnvironment, Technology

Introduction

Version control systems (VCSs) have a decades-long history in professional software engineering with early systems like Source Code Control System (SCCS) and Revision Control System (RCS) developed in the seventies and eighties respectively. These pioneering systems only supported storage of the versions on the file system, while later systems also allowed for remote and mostly centralized storage of the versions. The most well-known centralized systems are Concurrent Versions System (CVS) and Subversion (SVN). Currently, there is a trend towards the use of distributed version control systems (DVCS) where each user has a local copy of the repository which can be synchronized with other repositories. Systems such as Git and Mercurial exemplify this type of present-day decentralized technology. These DVCSs enable flexible change tracking, reversibility,

and manageable collaborative work, which are valuable for both small and large projects.

There are many arguments for incorporating VCSs into an educational setting. From a teacher's point of view, using VCSs increases the possibility of monitoring how students make progress with their assignments and eases the feedback process. The teacher could, for instance, include corrections and suggestions directly into the students' program code [1]. More generally, educators acknowledge that the use of VCSs relates to effective team work and that it is a crucial skill to be taught to prepare a competent workforce for present-day distributed workplaces [2].

An educational concern of interest to us is how students actually use VCSs. This has been previously studied by Mierle et al. [3] who investigated VCS usage patterns in a second-year course, hoping to find a correlation between an effective use of VCSs and study success. No clear patterns could be identified in the data which the authors attributed to the fact that beginner students climb their learning curve at different rates; see [3, 4]. These authors call for more research on VCS usage patterns in particular in upper-year courses [4], which motivates the present study.

We have collected data about students' use of the distributed version control system (Git) from three different courses: *Introduction to Software Engineering* (second-year bachelor), *Functional Programming* (master's level), and *Service oriented architectures and cloud computing for developers* (master's level). A hypothesis arising from teacher observations during these courses is that students use VCS principally as a submission system rather than what it is intended to be. By this we mean that

- students commit at the end of the class sessions or right before the deadline, or there is only one commit per week/task,
- only one group member commits everything,
- students do not consider what file types to commit,
- overall, with no specific training, student do not use VCS efficiently.

We study these issues quantitatively exploring version control commit frequencies, commit sizes and the activity of individual students. Our specific research interest is the potential usage patterns identifiable in the commit log data of Git repositories.

1 Version control systems in education

Clifton et al. [5] summarize that in educational settings VCSs have been adopted to enable more realistic software development experiences for students [6], as a tool to monitor or visualize team and individual contributions [7], and for non-code artifacts such as creative writing [8]. Clifton et al. themselves, as well as many others, use a VCS for course management purposes. Further, some authors regard VCSs as a valuable tool to monitor and understand *how* students develop code [9]. Unsurprisingly, one of the most usual educational targets appears to

be courses with project work where VCSs both foster team work and facilitate course management tasks such as assessment and grading [10]. Milentijevic et al. [11] go as far as to propose a generalized model for the adoption of VCSs as support in a variety of project-based learning scenarios. All in all, we find that there is a general consensus of the benefits of VCSs as an integral part of computing curricula, one key argument being that they measure up to the requirements of globally distributed workplaces [2].

There are also challenges in the educational use of VCSs. Reid and Wilson [4], who used the CVS system, report on the confusion in judging which of the students' assignment versions was the final one. Glassy [9] found that students tend to put off working on assignments for as long as possible, even though a VCS is proposed to them with the hope of iterative work processes. Issues of this kind relate to inefficient use of VCSs. Furthermore, Reid and Wilson [4] noticed that some students mixed the functionalities of the CVS check out and update commands, and that also teaching assistants encountered problems if they had not properly familiarized themselves with the tool. These issues were considered to be due to a lack of a mental model of the VCS system used. Yet another challenge Reid and Wilson [4] raise is increased teacher workload when repositories are initiated and managed by teachers. In a more recent study, Xu [10] points out that there can be a long and rough learning curve before students feel comfortable using Git. Accordingly, Milentijevic et al. [11], who used CVS, report that students find a VCS to be a useful tool after they are sufficiently familiar with it. In the paper by Glassy [9] and Xu [10], the value of informative commit log messages is raised as a topic to be emphasized to the students. Rocco and Lloyd [12] in turn observed that some student have difficulties in understanding what constitutes "a significant change" to be committed.

It is much more difficult to find systematic empirical studies on issues such as how frequently students make commits and how they share the work. Rocco and Lloyd [12] found in their data that over 80% of a CS1 course population could adopt an iterative work process with the Mercurial system (50.0% did 7–21 commits and 33.3% more than 21 commits). On another course the authors defined a minimum commit frequency for one assignment and no requirements for the assignment that followed. With the first assignment, 75% of the students obtained a reasonable commit frequency, while with the latter this was 81%, altogether indicating that informing students of proper VCS usage can have a positive effect on their work processes. 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.

The present study focusing on the students' usage patterns with the Git system in both a second-year course and master's level courses complements the studies such as the ones by Rocco et al. [12] and Mierle et al. & Reid and Wilson [3, 4].

2 The courses

Introduction to Software Engineering (SE) is a 3-credit course consisting of lectures, a course assignment, and an end-of-course exam. The lectures introduce students to the basic concepts of software engineering, while the mandatory course assignment is the preparation of a project plan. The assignment is done in small groups and consists of four larger phases that need to be accepted by the lecturer. 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. The course had altogether 72 students in 33 groups (2.18 ± 0.76 students per group).

Functional programming (FP) is a 6-credit course implemented without traditional lectures and exams. The course is run in week-long cycles such that each week a new set of exercises is announced for the students. Students work in small groups and all of their study time is devoted to programming the weekly exercises. Two contact sessions are held each week. The first one is devoted to supporting the students' work and answering their questions. During the second weekly contact session there is a review of the student-written code. Overall, the course emphasizes self-direction on the part of students, similar to recently discussed course models such as the flipped classroom; see more details in [13–15]. Git was proposed for students as their primary group work tool and all of the exercises had to be returned via it. Thirty-six students where active in the course divided over 13 groups. (2.77 ± 0.89 students per group)

The last master's level courses studied, *Service oriented architectures and cloud computing for developers* (SOA&CC), introduces students to the use of digital services and the concept of cloud computing. A format similar to the FP course is used during the first (5 credits) part of the course. During that part of the course students undertake independent group work on a set of assignments each week. Two weekly sessions are arranged for the group work and one mandatory contact session focusing on reflective program review is arranged at the end of each week. An analysis of how the course model used in this course attempts to motivate students can be found in [16]. During the course Git is not only used as a version control system; it is also used as a tool to deploy code to Platform as a Service (PaaS) providers. Nine groups of students were formed with altogether 36 students (4 ± 0.82 students per group).

All three courses utilize the Faculty's *YouSource*[1] system. Similar to staff members, students can use their university credentials to log in to this system and create projects and Git repositories to manage collaborative work. The projects and repositories can be defined to be either private or public and collaborators can be added to them with a variety of permissions. This system has been in use at the department since mid-2010 and has been used in many courses and research projects.

It should be noted that in the remainder of this paper we are specifically concerned with the Git version control system, which belongs to the third gen-

[1] `https://yousource.it.jyu.fi/`

eration of version control systems (DVCSs). Students are free in their choice of environment for interacting with the version control system. Students can for instance use the *git* command line tool, tools with a graphical user interface, or tools included in their integrated development environments.

3 Data analysis

The Git repositories which students or course teachers created for the respective courses on the above-mentioned *YouSource* system are the source of all data analysis in this paper. One limitation of this data source is that we cannot see any data related to branches which a student did not push to the central Git server. However, if work of one of these so called local branches got merged into a branch which is synchronized with the central Git server, we are able to see its history as well. Further, this limitation is of minor importance since we are mainly interested in how students use the version control system in group work settings. Another limitation, which is inherent to the Git DVCS, is that we cannot know for sure whether time stamps on commits are truthful. It is technically possible to tamper with the date of the commits, but since there is no benefit for students to do so, we make the assumption that the time stamps are correct.

To study our research hypothesis, we will perform five different analyses, the first four of which are based on commits to the repository and the last one on the content of the repositories. For each commit we extracted the number of insertions and number of additions in accordance with the short status log of each commit[2]. We added these two numbers together to form what we will call the number of changes of that commit. The tools used in the analysis have been developed by the authors of the paper and consume output produced by the diverse git commands.

For the first analysis we will, for each course, look at the commit activity over the whole course. To be concrete, we will visualize the commit activity by plotting the estimated probability density function of the total number of changes, i.e. for all students, over the span of the course. The density is estimated via the standard kernel density estimator, using a Gaussian kernel with bandwidth of 6 hours.[17] The height of the plot then shows the relative likelihood of a commit at a specific point in time.

The second analysis focuses on students' activity during the implementation sessions. This is done only for the FP and SOA&CC courses since the SE course does not have distinct sessions during which students get time to implement their work. We use a similar method as in the first part, but accumulate all commits that were made during the implementation sessions in the same plot. This plot shows when the students commit their code during the contact sessions. In the figure the far left of the x-axis represents the start of the session and the far right 15 minutes after the end. This is done in order to account for commits

[2] `http://www.kernel.org/pub/software/scm/git/docs/git-log.html`

right after the sessions. In this case we use a bandwidth which is one tenth of the total length of the session.

In the third part we perform an analysis of the commit messages in the different courses by classifying them in three categories : useful, trivial, and nonsense. A message is placed in the nonsense category if its content is not anyhow related to what is being committed. An example of this type of messages are these which contain only a couple of random letters, needed because the git system does not allow for empty commit messages. A trivial message is one which has no information beyond what is immediately visible from the commit meta-data. This category includes, for instance, a message consisting of a list of changed files or one saying that a given commit is a merge of two branches. All other commits are classified as useful. It should be noted that being in the useful class does not directly imply that the message is of high quality. It only means that the message is not trivial or nonsense. The classification was done manually by the respective teachers of the courses. We do not try to make a comparison between the courses, because the bias caused by having different raters is difficult to estimate.

In the fourth part we try to measure whether the version control system is used equally among the students in the group. If the system would be used by all students in a group, we would expect that the most active student in a group of n students performs $(1/n) * 100\%$ of the commits. To represent this number for all groups in the different courses we first find the students with the highest number of commits in their respective groups. Then we calculate their individual share in the total number of commits of their group. We then create an overview of the obtained percentages where we show different graphs for different group sizes since comparison among unequal group sizes would lead to biased results. It only makes sense to measure this for groups with more than one person. The SE course had a few single-person groups, hence only 28 groups from that course are included.

For the fifth and last part we investigate the types of files which students put under version control. First, teachers of each of the courses listed the file types and limits which they would expect a normal repository to contain. We started out from the files included in the HEAD of the master branch. For the FP and SOA&CC course we determined the type of each file using the BSD *file*[3] command. The SE repositories required a manual analysis to decide the type of the files because the *file* command is unable to distinguish between the file types in use in the course. Then we counted the number of files of each file type. Then for each count, we compared it to the number of files expected by the teacher and any surplus was counted as garbage. The final number which we calculated for each group is the fraction of garbage in the total number of files.

[3] `http://www.openbsd.org/cgi-bin/man.cgi?query=file`

4 Results

This section describes the results of our analyses. The first subsection shows the results for the analysis of the student activity during the whole course. In the second subsection, we focus on the implementation sessions only. The results of the analysis of the commit messages is shown in subsection three. Then we consider the activity distribution among students in the fourth subsection. Lastly, we look at the types of files which students submit to the version control system.

4.1 Commit activity over the whole course

The student activity in the SE, FP and SOA&CC courses is presented in the Figures 1, 2, and 3, respectively. In the SE course, we draw thick vertical lines to indicate the end of each of the four phases of the course assignment. As can be seen in that figure, there seems to be no correspondence between these deadlines and the student activity. In this course where VCS training was provided, students appeared to commit rather evenly throughout the course. We attribute the activity spike at the start of this course to the students trying out and getting familiar with the version control system at the point in time of the training sessions.

In the graphs of the FP and the SOA&CC courses (figure 2 and 3), we indicated with thin vertical lines the sessions during which students get time in the classroom to work on the assignments. The thicker vertical lines indicate deadlines for the weekly assignments. The dates of the sessions are displayed on the x-axis in a month/day format. In contrast to the SE course, these graphs show a closer correlation between student activity and the implementation sessions and the deadlines. The graphs suggest that most of the work was committed during the contact sessions, which again suggests that students bring their work to the sessions to be committed there. This prompts us to study the student behaviour during the sessions separately in the next section. It is also clearly visible that the students have a very low activity during the weekends.

4.2 Commit activity during the sessions

In the FP and SOA&CC courses students were more active during sessions than at other times. The graphs in figures 4 and 5 show the students activity during the sessions and 15 minutes after the session. We normalized the duration of the session (90 minutes) and the 15 minutes overtime between zero and one. Interestingly, we notice a similar behavior in both courses. There seem to be three periods of higher activity. The first moment of higher activity is in the beginning of the session after about 10 minutes. The second one, which last longer, is between 20 and 40 minutes after the start of the session and lastly, the activity peaks shortly after the session.

The first period of activity is most likely because individual students have been implementing parts at home. These students then decide to commit only

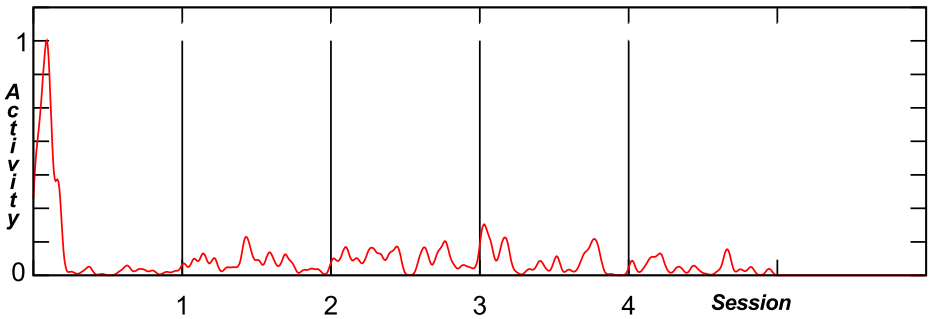

Fig. 1. Commit activity during the SE course

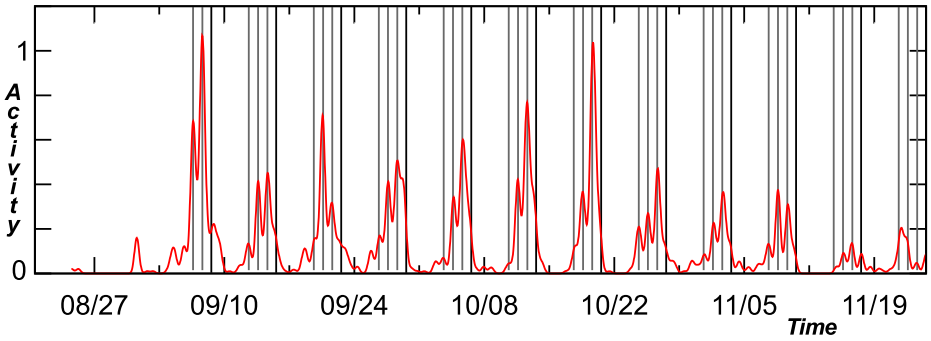

Fig. 2. Commit activity during the FP course

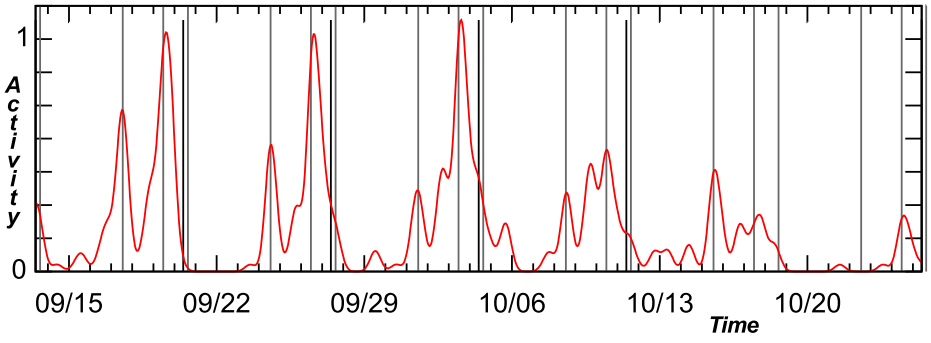

Fig. 3. Commit activity during the SOA&CC course

after receiving consent from other group members. This is an indication that students do not know how to use the version control system efficiently, as in principle they could have used a separate branch for their local development and merged their changes to their shared version of the exercises. Also, speculating based on student dialogue, some students might have feared 'losing face' by making their preliminary versions visible to others, including the teacher.

During the second period of activity students are using the system as they are supposed to, committing changes regularly. Then the activity drops for quite some time before reviving shortly after the session. We attribute this last peak to those groups who have been working during the whole session without committing many changes. At the end of the session they want to store their work for later continuation and decide to put all their work in the system.

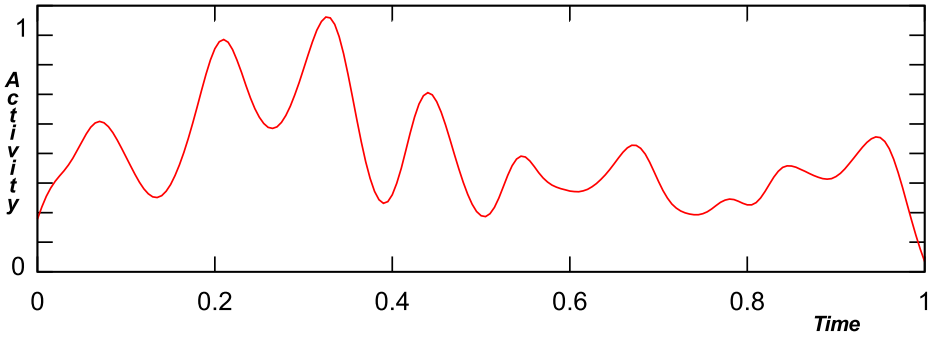

Fig. 4. Commit activity during the sessions of the FP course

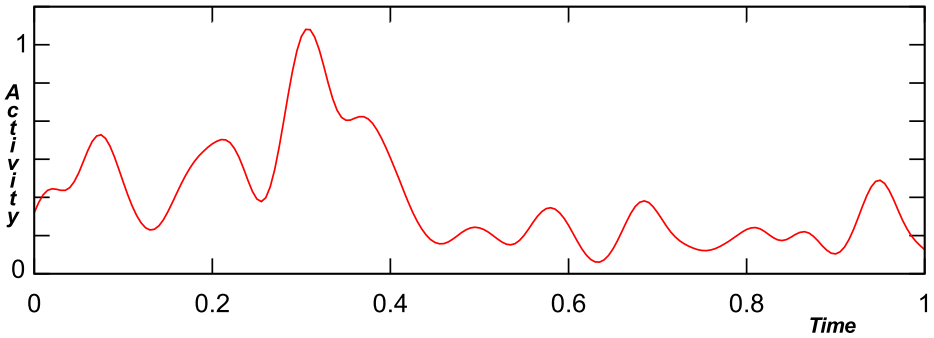

Fig. 5. Commit activity during the sessions of the SOA&CC course

4.3 Commit message analysis

The classification of the commit messages was performed for the SOA&CC and FP courses and yielded the results shown in table 2.

Table 1. Categorization of the commit messages per course

	useful	trivial	nonsense
SE	996 (67%)	430 (29%)	59 (4%)
FP	1422 (78%)	276 (15%)	129 (7%)
SOA&CC	289 (74%)	65 (17%)	37 (9%)

Table 2. Categorization of the commit messages per course

In an ideal repository we would not find any trivial and nonsense messages. What we see from the table however is that there is a significant amount of these types of messages.

It is not visible from the table, but the teachers classifying the messages shared the opinion that the messages in the useful category where not all that descriptive. Some commit messages could be regarded as 'locally sensible', meaning that they could be useful for communication during a short time span, but offer not much for later inspection. Many of the commit messages are clear indicators that the students regard the system as an answer submission system. Examples include "Answer for week 12" and "exercise 4a". We also noticed some messages related to problems in using the git system. The amount was however not as significant as the teacher had expected. It is also observed that the quality of the messages is depending on the group, indicating that some groups use the messages for communicating, while others do not.

4.4 Differences in student activity

To show the differences in activity among students we assembled the charts in figure 6. The figure contains one pie chart for each course and each group size or none if there are no groups of the given size in the course. Each pie chart illustrates the fraction of the groups which have a given percentage of commits for their most active committer. The last column shows the expected fraction, i.e. the chart which would be obtained if all students in the groups do an equal number of commits.

What we see from the charts is that the most active committer in a group, most of the times, commits significantly more as the expected percentage. Put another way, the most active committer in each group is very often much more active as the average which one might expect. This can be due to that student having a dominating role in the group. In the FP and SOA&CC course we tried to mitigate this effect by grouping students according to their skill level.[13, 15, 16] We however think that the main differences are caused by a different level

of familiarity with the version control system between the group members. The person with the most experience will commit more frequently or is given the task of submitting the work of others to the system.

4.5 Which files did the students commit to VCS?

The presence of files which do not belong in the version control system, such as executable programs, temporary compilation files and copied documentation, suggests that the students used the VCS as plain file storage. Figure 7 shows the fraction of the groups with a given percentage of redundant files in their final repository. We see a big difference between the figures for the respective courses.

In the graph for the SE course we see that most groups did not include many compiled files in their repositories, as was explicitly instructed in the course.

During the functional programming course, students can often test their code without actually compiling it, which could explain the low amount of garbage in the repositories. The garbage that is committed consists entirely of compiled binaries and other compiler generated files.

During the SOA&CC course many students use integrated development environments (IDE) which do the compilation automatically for the user. It seems like many students have included all files which the IDE produced to the version control system.

It seems that if students are not made aware of the fact that they should not include this kind of files to the VCS, they tend to include everything that happens to be present in their local directory. We should do further research to see whether this behavior changes if students are made aware of the best practices.

Conclusion

In this article, we focused on students' usage patterns in advanced courses related to programming while using the distributed version control system Git. We first looked at when students commit their work during the course and in more detail at their committing pattern during the implementation sessions. We concluded that most students commit changes regularly during the implementation sessions, but do not commit changes of work which they have been doing before the session itself. Some groups commit rarely during the session and make a big commit at the end of the session. We did some effort in classifying commit messages and noticed that students do often write messages which are either trivial or even sheer nonsense. Further, we looked at how the usage of the system is divided inside groups and found out that the activity of the most active user in a group is significantly higher than what would be expected if each group member would use the system equally much. As the last part of our analysis we considered the types of files which students put under version control. We concluded that if students are not told explicitly that they should not include certain types of files, they will just do so.

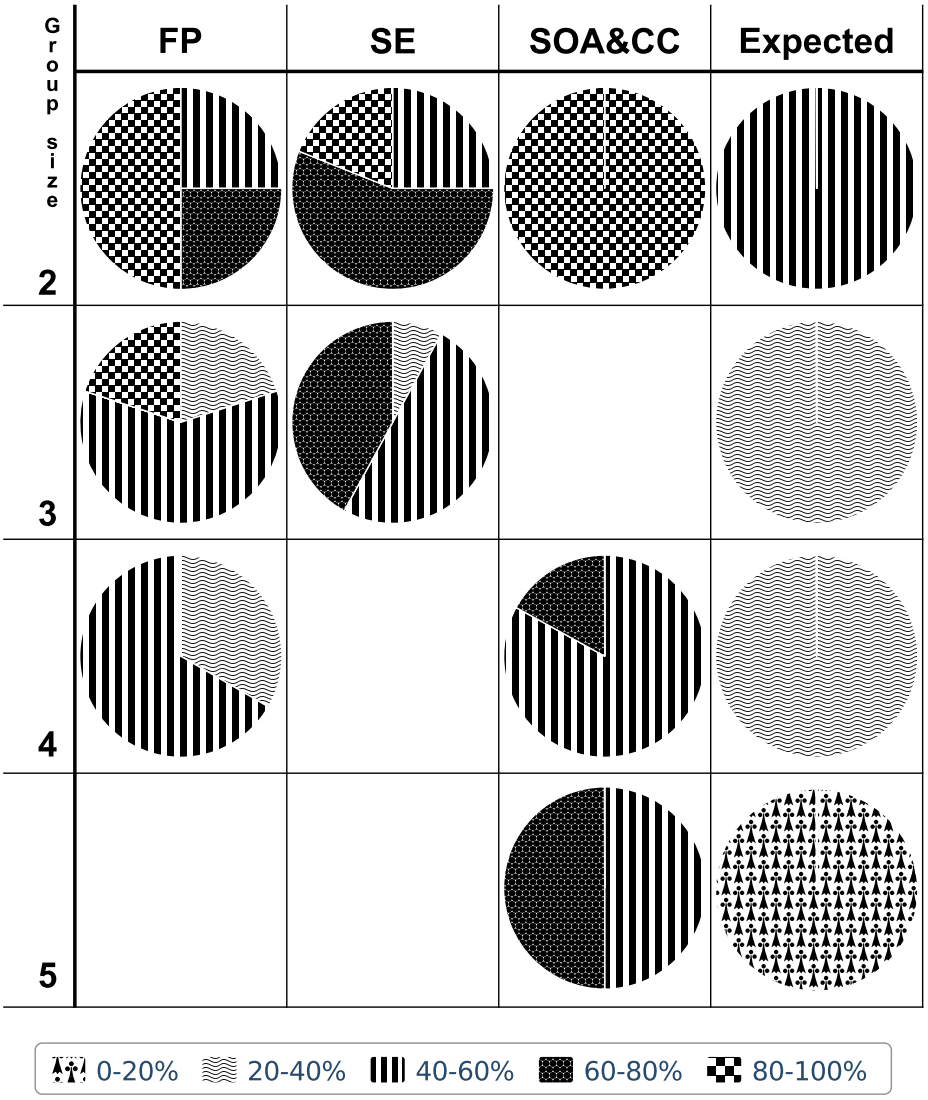

Fig. 6. Fractions of the groups with a given percentage of commits performed by its most active student

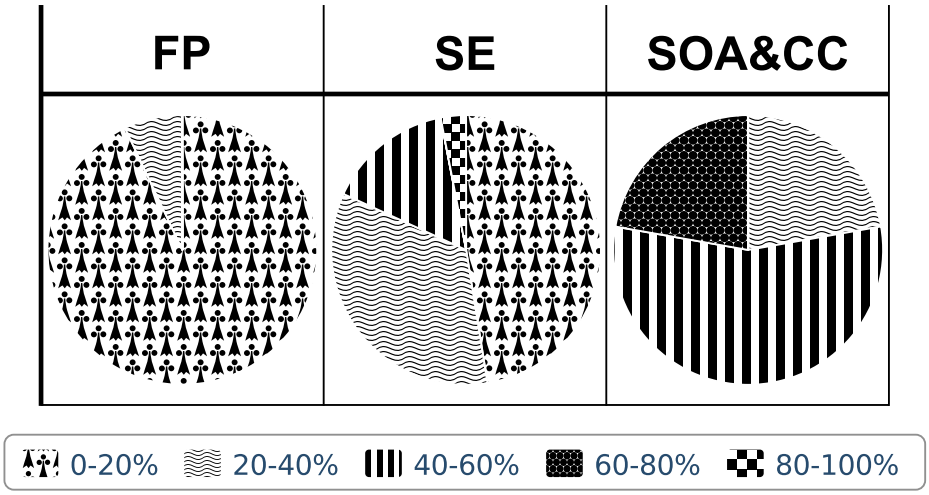

Fig. 7. Fractions of the groups with a given percentage of 'non-versionable' files in their repositories

With regard to the hypothesis put forward in the introduction, our findings suggest that using a VCS as a submission management tool may result in students adopting the tool as "just a required manner to return the assignments" — instead of a professional tool by which collaborative and distributed work is managed. Indications for this were that across all three courses studied the version control system was not too evenly used by the team members and the fact that the quality of the commit messages was quite low. While VCSs are pronounced as useful course management tools in the literature, we would like to note that professional use of VCSs requires support and demonstration of their usefulness.

In our future undertakings, we could make a distinction between submissions returns and use of VCS, and add VCS training to the beginnings of the courses. Further, the existing classroom setup where there is sometimes only a single computer for the whole group could be replaced with settings where each student would use a separate computer. Performing these practical changes could reveal whether a more intense and shared use of a VCS can be prompted among the students. Promisingly, in our second-year course, training sessions were provided and there were no observable commit peaks near the deadlines of assignment phases but a rather constant commit curve. Further research could also point out how effective the students can use the system and how the organization of the group work influences the use of the system. It might be that some students know very well how to use the system, but do not see any reason to use their skills up to a full extent in the given settings.

Acknowledgments: We would like to thank the department of Mathematical Information Technology of the University of Jyväskylä for both financial

and material support, without which this research would not have been possible. We would also like to thank the reviewers for their useful corrections and suggestions, which greatly improved the quality of this paper.

References

1. Laadan, O., Nieh, J., Viennot, N.: Teaching operating systems using virtual appliances and distributed version control. In: Proceedings of the 41st ACM technical symposium on Computer science education. SIGCSE '10, New York, NY, ACM 480–484 (2010)
2. Meneely, A., Williams, L.: 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, New York, NY, ACM 529–533 (2009)
3. Mierle, K.B., Roweis, S.T., Wilson, G.V.: CVS data extraction and analysis: A case study. technical report utml tr 2004-002. Technical report (2004)
4. Reid, K.L., Wilson, G.V.: 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, New York, NY, ACM 272–276 (2005)
5. Clifton, C., Kaczmarczyk, L.C., Mrozek, M.: Subverting the fundamentals sequence: Using version control to enhance course management. SIGCSE Bull. **39**(1) 86–90 (March 2007)
6. Hartness, K.T.N.: Eclipse and CVS for group projects. J. Comput. Sci. Coll. **21**(4) 217–222 (April 2006)
7. Liu, Y., Stroulia, E., Wong, K., German, D.: Using CVS historical information to understand how students develop software. In: MRS 2004: International Workshop on Mining Software Repositories. (2004)
8. Lee, B.G., Chang, K.H., Narayanan, N.H.: An integrated approach to version control management in computer supported collaborative writing. In: Proceedings of the 36th annual Southeast regional conference. ACM-SE 36, New York, NY, ACM 34–43 (1998)
9. Glassy, L.: Using version control to observe student software development processes. J. Comput. Sci. Coll. **21**(3) 99–106 (February 2006)
10. Xu, Z.: Using git to manage capstone software projects. 159–164 (2012)
11. Milentijevic, I., Ciric, V., Vojinovic, O.: Version control in project-based learning. Computers & Education **50**(4) 1331–1338 (2008)
12. Rocco, D., Lloyd, W.: Distributed version control in the classroom. In: Proceedings of the 42nd ACM technical symposium on Computer science education. SIGCSE '11, New York, NY, ACM 637–642 (2011)
13. Tirronen, V., Isomöttönen, V.: Making teaching of programming learning-oriented and learner-directed. In: Proceedings of the 11th Koli Calling International Conference on Computing Education Research. Koli Calling '11, New York, NY, ACM 60–65 (2011)
14. Tirronen, V., Isomöttönen, V.: On the design of effective learning materials for supporting self-directed learning of programming. In: Proceedings of the 12th Koli Calling International Conference on Computing Education Research. Koli Calling '12, New York, NY, ACM 74–82 (2012)

15. Isomöttönen, V., Tirronen, V.: Teaching programming by emphasizing self-direction: How did students react to active role required of them? ACM Transactions on Computing Education Research (accepted)
16. Isomöttönen, V., Tirronen, V., Cochez, M.: Issues with a course that emphasizes self-direction, submitted. (2013)
17. Parzen, E.: On estimation of a probability density function and mode. The annals of mathematical statistics **33**(3) 1065–1076 (1962)