Version Control Usage in Students' Software Development Projects

Pekka Mäkiaho, Timo Poranen and Ari Seppi

Abstract: *The need for a version control system rises always when multiple developers work on the same project and is beneficial even for projects of one developer. The core tools of computer science curriculum also include version control systems; therefore it is essential to teach the usage of such tools to all computer science students in universities. In this paper we study the usage of version control in students' software development projects. Data was gathered from 13 different student project teams. We analysed challenges and main ways to use version control in training, development and management. We noted that the understanding of main principles of using version control is not always clear to students and this should be taken in account in teaching. There were also huge differences between teams in the number of commits (from 32 to 359). Low number of commits might be a warning signal of problems in tool usage and in project in general.*

Key words: *Software development projects, university courses, version control usage.*

INTRODUCTION

Software development project courses are common in all universities. In a typical project a student team implements a software product using some widely known software development model. In ACM's and IEEE's computer science curriculum 2008 [1] recommendations this is explained as follows:

Significant project experience. To ensure that graduates can successfully apply the knowledge they have gained, all students in computer science programs must be involved in at least one substantial software project. Such a project (usually positioned late in a program of study) demonstrates the practical application of principles learned in different courses and forces students to integrate material learned at different stages of the curriculum. Student need to appreciate the need for domain knowledge for certain applications, and that this may necessitate study within that domain.

Development tools also play an important role in computer science education and in industrial and global software development projects [10]. ACM's and IEEE's curriculum [1] lists the following tools and categories as a core: programming environments, requirements analysis and design modeling tools, testing tools including static and dynamic analysis tools, tools for source control, and their use in particular in team-work, configuration management and version control tools and tool integration mechanisms.

The need for a version control system (VCS) rises always when multiple developers work on the same project and is beneficial even for one developer projects. The advantages include possibility to work together but separately on the same piece of code, having multiple branches (separated distributions of the application), investigating what change caused a bug and how the application used to work before the change and simply expunging the code that is later found to be invalid.

In this paper we study version control usage in students' software development projects. First we give a deeper introduction to version control systems and then we describe how data was gathered and analyzed. Finally we conclude our work and discuss future research directions.

CompSysTech'14, June 27-28, 2014, Ruse, Bulgaria.
Copyright ©2014 ACM ISBN 978-1-4503-2753-4/14/06...$15.00.
http://dx.doi.org/10.1145/2659532.2659646

VERSION CONTROL

The currently used VCSs are commonly [3] divided into two generations based on their repository centricity: client-server architecture and distributed storage. Some sources also refer to the first generation of local-only systems as a historical curiosity [11].

The first generation of these local-only storage systems was created in the 1970's and 1980's. The second generation of centralized client-server solutions began to appear in during the 1980's, the two milestones being CVS (1986) [5] and its successor Subversion (2000) [12]. The development of third generation distributed systems started in the 1990's, but the current big names Git [6], Mercurial [8] and Bazaar [4] were first released in 2005. In the distributed VCSs each user has their own repository to which they then merge changes from other repositories.

In addition to the storage approach, the main difference between the generations is in the way the concurrency is handled in each generation's version control systems. The first generation systems usually handled the problems by forcing a lock on a file before editing it, the second generation employed a merging mechanism on a commit and in the third generation systems commit to the local repository takes place before the merge; merging takes place when the changes from other repositories are integrated to the local repository. Of course, the lines between the generations are not strict so there are also lock-based client-server version control systems.

The four most commonly used VCSs are CVS, Subversion, Git and Mercurial. CVS, which was once the standard version control system, is now mostly used in legacy projects. The leader on the field is Subversion, but more clever third generation systems have advantages over the second generation [7] and Subversion is steadily losing ground especially to Git. For open source projects one of the most important advantages is the ease of commit management [3].

In Subversion the workflow starts as a new developer begins working with the project by downloading the sources from Subversion repository (Subversion command *checkout*). After the developer has made a change to their local version they wish to include to the release, they send (*commit*) the modified source to the Subversion repository where the system replaces the previous version with the committed files. If another developer has modified the file in between the check out and the commit, the server returns an error message.

In case of such error, the user can usually fix it simply by performing an *update*-operation for the file, in which the system then merges the changes from the server to the local file, which was modified by the developer. If the files have been edited on different parts, this process is automatic. If not, the user has to solve the conflicts on the client. After the merge and possible conflict solving the developer can successfully commit the file to the server.

Release management in Subversion is handled by creating a branch which is a copy operation in the Subversion file system. Each project has *trunk* and *branches* directories, the main part of the development takes place on the trunk and as a project is released, a branch is created by copying the *trunk* as *branches/release version* directory.

DATA GATHERING

At the University of Tampere, School of Information Sciences, Project Work [2] is a compulsory BSc.-level course to all 3rd year students who study computer sciences. The course is organized yearly and the projects last almost seven months including season break in the New Year. Project teams have 4-5 developers from Project Work course and several project managers from MSc.-level course Software Project Management.

All students participating the course have studied at least 50 ECTS basic computer science and interactive technology courses (programming, databases, user interface design and evaluation, information systems and software development). The managers have studied approximately one year longer than Project Work students, including at least one project where they participated as developers.

Table 1: Overview of projects.

Type	Development model	Version control	Commits (w. 11)	Used tools
Android	Scrum	subversion	79	Eclipse, rapidSVN, TortoiseSVN, command line
Android	Waterfall	Subversion	208	Eclipse, TortoiseSVN, command line
Appl.	Scrum	Subversion	38	MS Visual Studio, Eclipse, TortoiseSVN
WWW	Scrum	Subversion	205	MS Visual Studio, TortoiseSVN
WWW	Scrum	Subversion	71	Editplus, TortoiseSVN
Appl.	Scrum	Subversion	32	MS Visual studio, TortoiseSVN
WWW	Scrum	Subversion	204	Eclipse
WWW	Scrum	Git	359	NetBeans IDE, command line
Andr.	Scrum	Git	306	Eclipse, command line, GIT for Windows,
WWW	Scrum	Subversion	153	TortoiseSVN, command line
Appl.	Scrum	Subversion	125	Eclipse, TortoiseSVN, command line, RabbitVCS
Appl.	Scrum	Subversion	206	MS Visual studio, TortoiseSVN, command line
WWW	Scrum	Subversion	103	NetBeans IDE, TortoiseSVN

In the beginning of the project work course, there is a lecture on tool usage in projects. The lecture introduces basics on wikis, project management tools, UI-design tool and Subversion version control. Currently there are no hands-on exercises for these tools, instead the students are encouraged to organize training with their project team if they need help with the tools.

The university provides all project teams a subversion repository, but the teams have freedom to use any other version control tool if they like. The teams also have the freedom to select their development model, although course's recommendation is to use iterative and agile development models. During the course there are several checkpoints, where the course staff participates: preliminary analysis meeting, project plan review, three iteration reviews and the final meeting. Even though, there are only three iteration reviews with the course staff, all the teams should have more iteration reviews with their clients depending on the needs and scheduling of the project.

During the academic year 2012-2013 there were 54 students taking Project Work course and 39 students participating in Software Project Management course. Course staff formed 13 project teams from the participants, the most common team size was 3 managers and 4 developers. The projects started in week 37 in 2012 and most of them ended in week 11 in 2013, only one project needed extra time.

All projects had real clients from university, industry or from non-commercial associations. Six of these projects developed www-applications, three Android-applications and four stand-alone applications which used Microsoft Kinect. Two projects developed further an existing software. An overview of projects is shown in Table 1 with information on project type, development model, used version control system, number of commits in the end (week 11 in 2013) and used version control related tools. It should be noted that even 12 projects reported they used Scrum, the right term would be Scrum-but, because students had no real chances to work daily together.

The data was gathered using Moodle questionnaire and some issues were also verified from a public project report [9] and weekly reports. In the beginning of January 2013, four months after the beginning, the following questions were asked from all participants:

Q1.1. What are the version control software and clients that you use in your project?

Q1.2. What kind of earlier experience (before this university project) you had on using version control?

Q1.3. What is your "main role" in the project and how many commits (roughly) you have committed to the version control so far?

Q1.4. Describe how you personally use the version control in your project. List also functionalities you have used (like: update, commit, add, status, branching, tagging, etc.)

Q1.5. What kind of problems you have encountered when using version control?

Q1.6. If you have any other comments on version control usage, you can tell them here.

The following questions were asked only from project managers.

Q1.7. As a project manager, how you have planned / managed / trained version control related activities in your project?

Q1.8. Do you have some kind of releasing schedule? How do you accomplish the interim and final releases (either with or without source code control)?

Q1.9. What kind of difficulties (from project management point of view) you have met when using version control?

After projects finished in March (about seven months after beginning) the following questions were asked from all participants:

Q2.1. How many commits you did during the project? What was the total amount of commits in your project? Your answer should contain just two numbers, like 0/123 or 57/123)

Q2.2. Describe how you personally used the version control (after new year to the end of the project). Were there any changes in the usage compared to your previous answer (Q1.4)?

Q2.3. What kind of new problems, if any, you encountered in the version control usage (after new year to the end of the project)?

And the following two questions were asked only from project managers:

Q2.4. Did you have to change working practices (or guidance / management / tools etc.) of your team after New Year related to version control usage? If yes, explain.

Q2.5. When thinking about version control usage from project management point of view, what kind of things are the most challenging?

To answer these questions was a compulsory part of the course, so the answer rate was very high, only couple of students did not answer at all or gave empty answers to some questions.

VERSION CONTROL USAGE IN PROJECTS

Answers for each question were first anonymized and then grouped by corresponding project. To analyze the answers, they were further grouped into following themes: The used tools and version control systems, Earlier experience and training (Q1.2, Q1.7), Number of commits during the project (Q1.3, Q2.1 and data from weekly reports), Version control usage and problems (Q1.4, Q1.5, Q2.3) and Project management and version control (Q1.8, Q1.9, Q2.4 and Q2.5).

THE USED TOOLS AND VERSION CONTROL SYSTEMS

Only two projects used Git and 11 projects used Subversion repository provided by the university. A vast majority of the projects using Subversion on Windows platform used TortoiseSVN as a client. The second most popular tools were Eclipse's SVN plugins (Subclipse and Subversive), some users used SVN command line tools and there were single users with Visual Studio plugin and RapidSVN Linux client. Git project members preferred the command line client.

EARLIER EXPERIENCE AND TRAINING

Out of the 13 groups all had at least one member who had no experience on VCSs and at least two members who did. 27% of the students, who answered the question had no experience with VCSs, 52% had used VCSs on university projects or on their own projects and 21% reported on having industrial experience in using VCSs.

The training to non-experienced members were given in various ways: most of the groups did not have any formal training but the help was given on need basis. Two groups had written instructions.

The level of the training also varied. One group started the project just by helping and explaining, but in the middle of the project this was noticed to be insufficient and written instructions and a formal process were created. On the other hand, in one group members were taught to commit to personal folders and project managers merged the code.

NUMBER OF COMMITS DURING THE PROJECT

The project groups started working with version control between weeks 41 and 47 (Figure 1 and Figure 2). Mostly the flow of commits continued with roughly at constant volume after that and most of the projects had a noticeable increase in weekly commits after the turn of the year. This increase was especially dramatic for projects 4 and 8.

A curious exception to the turn of the year behavior was the project 9 that was clearly the most active version control user until the turn of the year, but very little happened after that because the software was almost ready. Another exception is the project 2 that started the last (week 47) but had a very high number of commits at the first week of work and after that continued having a commit frequency comparable to the other active projects.

The average amount of the VCS-commits on a project was 137 and it varied between 30 and 360. Out of the 78 students who answered the question about the number of the commits, the average was 23. 19 students reported zero. The maximum number of commits per student was 180. Naturally, the amount of the commits depends on a person's role on the project. UI-designers did zero or a few commits as the persons who focused mainly on coding reported much higher numbers. The average number of commits per a project manager was 15 and among of the developers (counted all non-project managers) it was 22.

The numbers of commits don't always correlate the lines of code written: there were reported quite creative ways of using VCS like sharing code by email or memory stick to one person who took care of the committing.

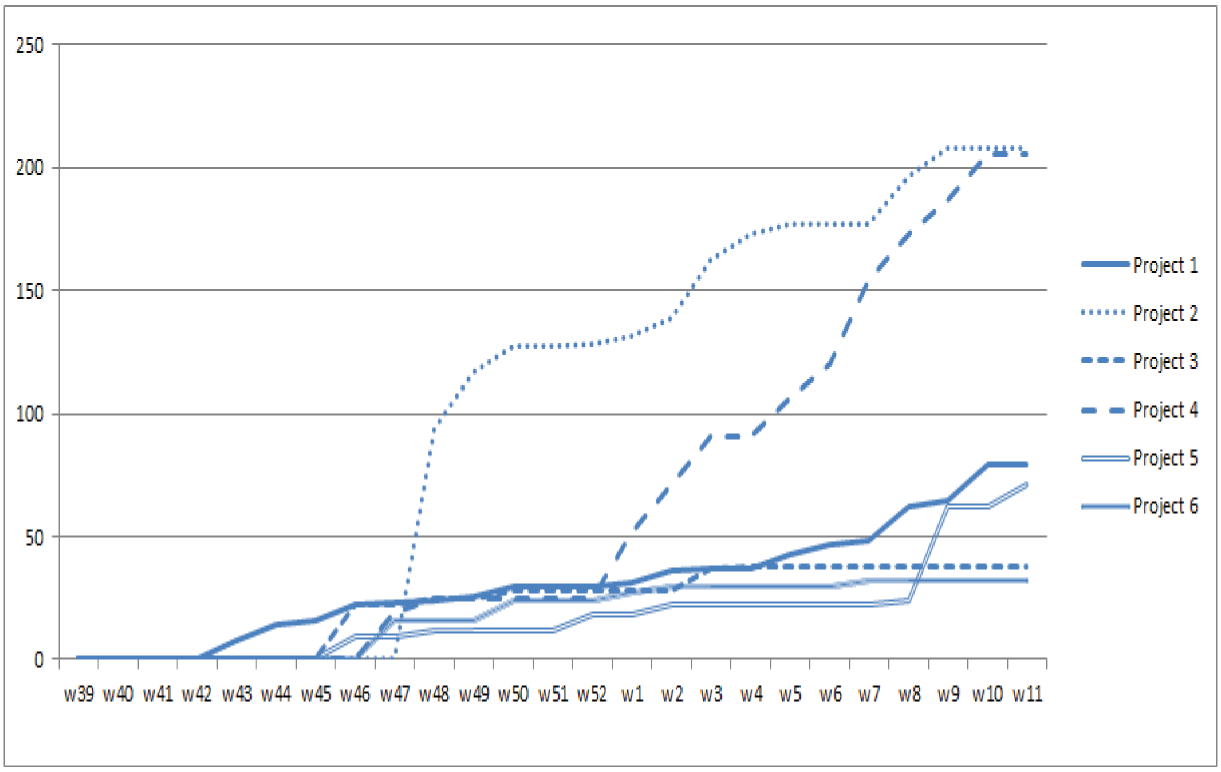

Figure 1. Version control commits of projects 1-6.

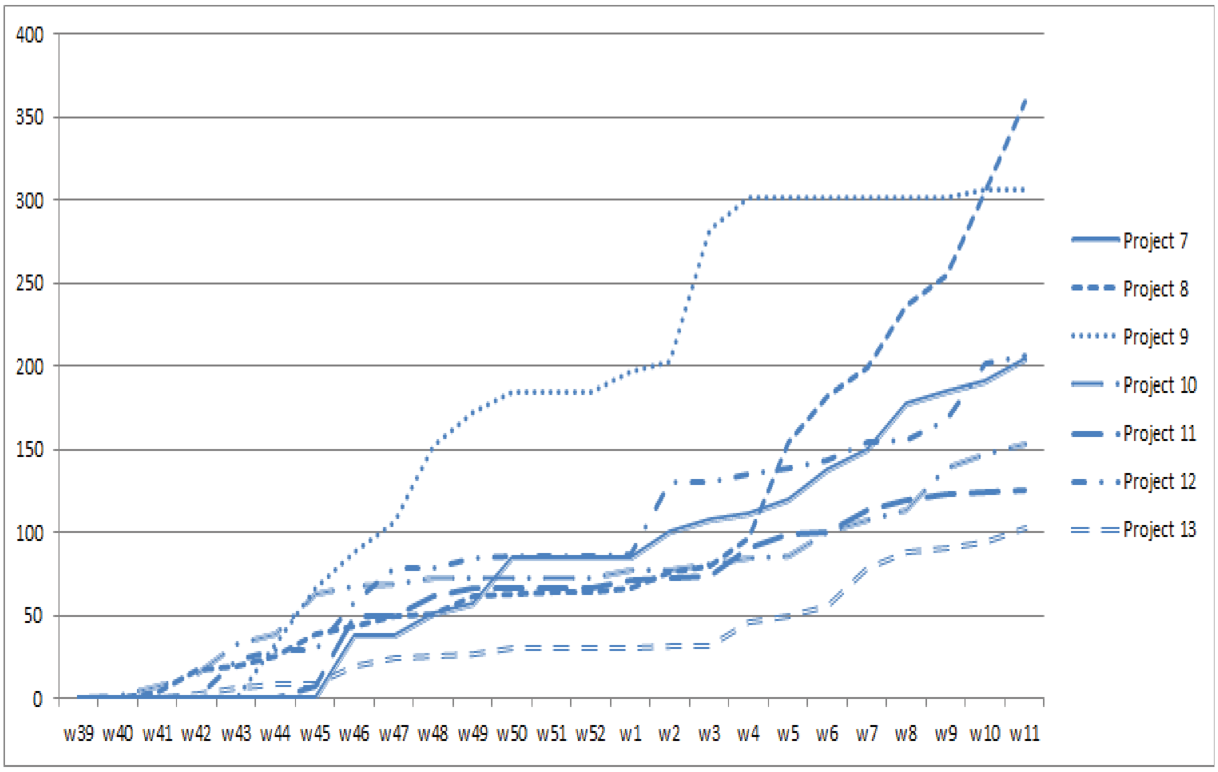

Figure 2. Version control commits of projects 7-13.

VERSION CONTROL USAGE AND PROBLEMS

In addition to the basic functionalities of add, commit and update, two groups mentioned using ignore command. Six groups had notions about status command and five groups mentioned using branching.

There were little changes in version control usage during the project. The groups that observed changes at all only mentioned committing more frequently in the later stages of the project.

The groups were asked about the problems in the VCSs usage in the midway of the projects and in the end of the projects. The reported problems were divided into three categories: *familiarity with the chosen tool*, *understanding the usage principles* and *problems with the environment*. When asked about the problems with the VCS in the middle of the projects, setting up the environment was mentioned as a problem by 7 groups. This was expressed with different sentences like "could not connect to the server" or "setting up the environments to everyone took a huge amount of time".

Students were neither familiar with the tools they selected; this was also reported by many ways from blaming the tool to complaining about flat learning curve. Not familiar with the tool was reported by 11 groups out of 13.

As 27% of students had no experience on VCSs, one might assume that the lack of knowing VCS principles would have been a problem. However, it was here reported only by 3 groups. After the project, 7 groups reported no problem or just minor problems like client handling Scandinavian letters in an odd way.

Three groups had problems with their environment on which one group member reported that he had to reinstall the system. Actually this group had also many other problems: in the middle of the project they reported problems from all the three categories, moreover, they blamed e.g. on “missing Macintosh-support”.

Problems on using the tool (not familiar with the tool) were still reported by 5 groups (11 in the beginning). Missing knowledge of VCSs principles were reported by one group after the project (2 on the middle of the projects).

PROJECT MANAGEMENT AND VERSION CONTROL

Most of the groups did not have any specific releasing schedule, their plan was simply to release when the application was done. There was one exception of a project that worked in sprints and provided interim releases for the customer after each sprint.

Common problems were related to lack of experience with VCS in general. More specifically, project members did not update their workspace from the VCS regularly, committed their own code rarely and as a combination of the prior two projects ended up with update conflicts and difficult merge problems causing one group even to drop VCS usage altogether. Even the groups that did not have problems of this scale had difficulties in getting the project members to remember to write commit comments.

Most of the groups did not report having any work practice changes. For those that did the changes mostly had to do with more rigorous version control guidelines and more instructions.

CONCLUSIONS AND FUTURE WORK

Our study concentrated on finding possible problematic aspects of version control usage with inexperienced student project teams. It is quite clear that for a software development team some kind of support for learning and using new tools should be provided by the team itself or by the teachers in university courses. We also were able to categorize version control usage problems to three categories: unfamiliarity with the tool, not understanding usage principles and problems with the environment.

One interesting finding was that there are huge differences in the number of commits between the teams. We noted that the teams that reported many problems and used version control in a non-conventional way had less commits than other teams. Our sample was quite small and also we did not perform any deeper analysis to detect connection between number of commits, real productivity (implemented functionalities, code lines), and quality. This is one direction that needs further research.

Also it is normal that projects have challenges with new technology and tools. We did not analyze if the teams that reported problems with version control also had other problems (like communication, motivation, weak skills, etc.). It is possible that low amount of commits is a possible warning signal on other problems that need manager's attention.

REFERENCES

[1] ACM and IEEE joint task force. Computer Science Curriculum 2008.

[2] Ahtee, T. and Poranen, T. Teaching software projects in universities at Tampere. In Proceedings of INSPIRE XII: Improving Quality in Computer Education. Pages 87-101, 2007.

[3] De Alwis, B.; Sillito J. In: Proceedings of ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE'09), 2009.

[4] Bazaar version control system. http://bazaar.canonical.com. 2014.

[5] Concurrent Versions System. http://savannah.nongnu.org/projects/cvs. 2014.

[6] Git - distributed version control system. http://git-scm.com. 2014.

[7] Malmsten, C.F. Evolution of version control systems - Comparing centralized against distributed version control models. University of Gothenburg, Report No. 2010:017, 2010.

[8] Mercurial.selenic.com. Mercurial source control management. 2014.

[9] Mäkiaho and Poranen, T. Software Projects 2012-2013. University of Tampere, School of Information Sciences, Report 23 (2013).

[10] Portillo-Rodriguez, J., Ebert, C., Piattini, M., Vizcaino, A. Tools to support global software development processes: A survey. In: Proceedings of International Conference on Global Software Engineering 2010., pages 13-22.

[11] Raymond, E. Understanding Version-Control Systems (draft), http://www.catb.org/~esr/writings/version-control/version-control.html. 2014.

[12] Subversion version control system. http://subversion.apache.org. 2014.

ABOUT THE AUTHORS

PhD student Pekka Mäkiaho, MSc, School of Information Sciences, University of Tampere, Finland, E-mail: pekka.makiaho@uta.fi.

University lecturer Timo Poranen, PhD, School of Information Sciences, University of Tampere, Finland, E-mail: timo.t.poranen@uta.fi.

PhD student Ari Seppi, MSc, School of Information Sciences, University of Tampere, Finland, E-mail: ari.seppi@uta.fi.