Using CVS Historical Information to Understand How Students Develop Software

Ying Liu, Eleni Stroulia, Kenny Wong
University of Alberta
Edmonton, Alberta, Canada
{yingl, stroulia, kenw}@cs.ualberta.ca

Daniel German
University of Victoria
Victoria, BC, Canada
dmg@cs.uvic.ca

Abstract

Software engineering courses are expected to teach students a wide range of knowledge and skills, e.g. software-development methodologies, tools, work habits, collaboration skills, a good sense of scheduling, etc. In this paper, we present a method to track the progress of students developing a term project, using the historical information stored in their CVS repository. This information is analyzed and presented to the instructor in a variety of forms. The goal of this analysis is, first, to understand how students interact, and second, to find out if there is any correlation between their grades and the nature of their collaboration. Understanding these factors will enable instructors to detect potential problems early in the course of the students' projects, so they can concentrate their help on those teams who need it the most.

1. Introduction

During an undergraduate software-engineering education, students are expected to become knowledgeable in advanced software-development methods, acquire good time-management habits, and learn effective collaboration skills. In this paper, we discuss a set of analyses that support the monitoring of student teams and their progress, based on information collected from their CVS [5] project repositories. These analyses infer a multi-perspective trail of the project development, and a set of corresponding visualizations presents various statistics, diagrams, and reports on this trail. Based on the information produced, instructors can track the evolution of a team's work against other teams or compare the performance of members within a team. Furthermore, instructors can inspect the revisions to an individual file or the modification requests of the project. (A modification request or MR is a set of revisions that is considered atomic and results from a CVS commit command [1]). We believe that if this information is suitably presented and highlighted, it could also be useful to students to self-evaluate their own progress.

This paper begins with a description of the directly collected and derived data examined by our method. Section 3 discusses our case study and highlights some initial experimental results. Section 4 outlines the related work. Section 5 concludes the paper and discusses some future work.

2. Collected data

Useful data is captured in the CVS logs. Examining the information implicit within them about development processes and software trails is crucial for monitoring and managing a software project.

2.1 Directly collected data

A substantial amount of information can be extracted by examining the data directly collected by the CVS repository. Trends in this data can be inferred and presented through diagrams or reports, leading to meaningful insights regarding the development of the team projects.

2.1.1 The *team* level

With the same project requirements, comparisons across teams are very useful for instructors to monitor the performance of the whole class, and quickly notice unusual trends and events that might signify problems. For each team, we record the following parameters: total number of files, total number of Java files, total number of MRs, total number of revisions, average CVS operation distribution by type [2] and date, average work days, average work days on Java files, the proportion of MR size, and so on. Diagrams based on these parameters are introduced in the next section.

2.1.2 The *individual-developer* level

Within a particular team, we need to look into each member's contribution, and suggest adjustments if necessary. We also ask each student to complete a questionnaire to describe their own contributions and what they perceive as the contributions of their team members. For each member, the following parameters are gathered: number of total CVS operations (of each type), number of modified files, number of modified Java files, number of added files, number of added Java files, and number of Java files last modified, total number of added and deleted lines of code (LOC), total number of work days, total number of work days on Java files, the first checkout date, the first file add date, the first file modification date, the last file modification date, the last operation date, self and peer assessment questionnaire data, and scores achieved on the project stages. Using this data, instructors and team members can become aware of

the work habits of individual students, their workload, and any problems that need to be addressed.

2.1.3 The *file* level

The above parameters enable the comparative analysis of individual students' work. To discover potential problems with the project design and the task division, more data about the project files themselves is relevant. For example, two potential problems may be files that have a high occurrence of colliding changes, or files that end up being modified by multiple members. For each file, the following parameters are gathered: final size, log size, number of revisions, number of individuals who modified it, total numbers of operations of each type, and added and deleted LOC of each member.

2.2 Derived data

In addition to the information captured in the CVS history, there still exist underlying relationships between the team members' work habits, their roles, their main tasks, the project-design structure, and project schedule which are implicit in this data. Examples of further analyses that could shed some light on the above relationships are the following: the proportion of each student's idle days to project duration, the proportion of a student's leading idle days (between the start of the project and the student's first CVS operation) to the entire project duration, the proportion of the tailing idle days (between the student's last CVS operation and the end of the project) to the entire project duration, the proportion of Java files to the whole project, and the proportion of various types of CVS operations on Java files to all CVS operations.

3. The case study

To evaluate the usefulness of the information discussed above as a means of monitoring project progress, we conducted a case study, in the context of a third-year software engineering course, in which students work on a project in teams of four. This work was originally done in the context of our JRefleX project [3,4] (http://www.cs.ualberta.ca/~stroulia/JRefleX). Here, we further analyze the collected data to include information collected at the MR level [1] and compare the inferences of our analyses against the students' own understanding of their project work.

In the context of this course, students coordinate their software changes using CVS. The project is common across all teams, with three delivery dates spanning a two-month development period. Although various deliverables are required, including unit test cases, UML diagrams, and a user manual, we initially focus our analysis on changes to the source code over time. Detailed information can be seen in [2].

This section introduces selected charts and statistics generated in our case study on five student teams (labeled A to E). Diagrams can be presented at various levels: by course, by team, by individual, or by file. Such diagrams intuitively show trends, enabling the users to gain a high-level impression of team and individual performance. One aim is to notice anomalies, such as delayed or unbalanced workloads. Statistical reports list the data gathered in our database for more detailed inspection.

3.1 MRs and CVS operations for all teams

Figure 1 shows the number of MRs over time. Figure 2 compares the average number of CVS operations by type across teams. A comparison of team operation numbers across time can also be generated, although not shown here for the sake of space.

The aim of these diagrams is to compare the various work habits of the student groups. How fast do they start? How long is their actual development process? How many idle days do they have? How many files do they work on at a time? What proportion of files are Java? What is the distribution of their CVS operations? Considering these diagrams, we observe that group D has the most CVS operations (such as file additions and modifications), has more regular workload habits than the other teams, and has a medium number of MRs. Groups B, C, and E have sharp peaks around each delivery deadline, preceded by long idle periods. Group B has the most MRs at the second deadline. Group C has the smallest number of MRs.

3.2 CVS operations and file-related information for individuals

For a specific team, three kinds of diagrams and two kinds of reports can be generated. One diagram (see Figure 3) compares across team members the number of CVS operations over time. A second diagram compares across team members the numbers of CVS operations of each type (see Figure 4 for group D). A third diagram displays, for all the files by a given team, the proportion of added and deleted lines of code by the team members (see Figure 5). The daily report lists all the history logs in the CVS repository by date chronologically, while the individual report shows all CVS operations by each member chronologically.

The purpose of these diagrams and reports is to compare workloads and work habits of individuals within a team. For example, we want to discover who contributes what in which role, such as whether there are students who write stubs to be filled in by others. Are certain members focused on testing and debugging? Or, are all members assigned a full variety of development tasks?

3.3 CVS operations and modifications at the file and revision level

Two file-level diagrams (one shown in Figure 5) and one file-level report are intended to display the information related to all the files being worked on by a team. Users can notice the high collision files and files modified by multiple members quickly in the diagrams. The file-level report lists the detailed history of CVS operations on a

file in time order. Many questions are relevant at this level. What are the number and types of files that a team has? How many files are there? What is the proportion of heavily modified files? How many times is a file modified or touched? Who is the author of a file? Is the author the last person to modify a given file? How does the number of files evolve over time? What is the distribution of CVS operations on each file? Which files belong to the core of the software? Which files should be in an independent module?

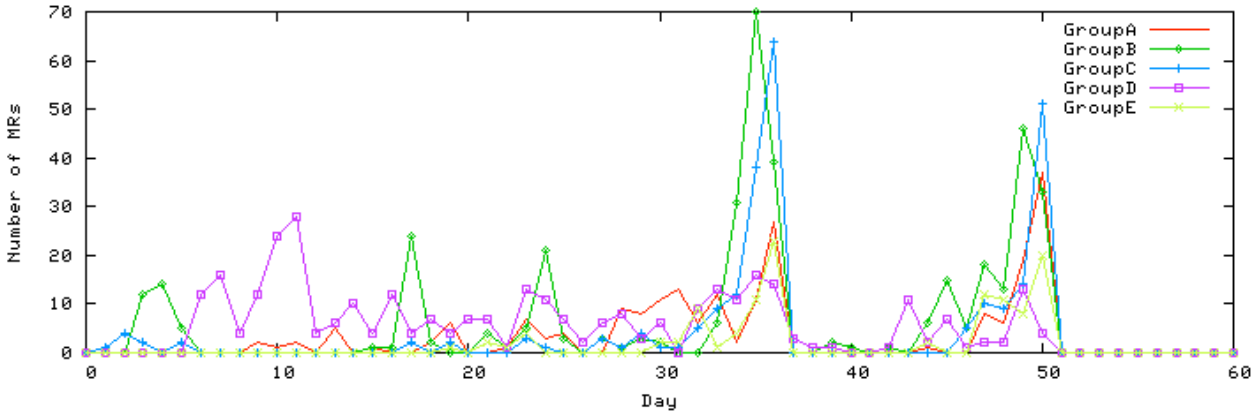

Figure 1: Modification requests of the teams

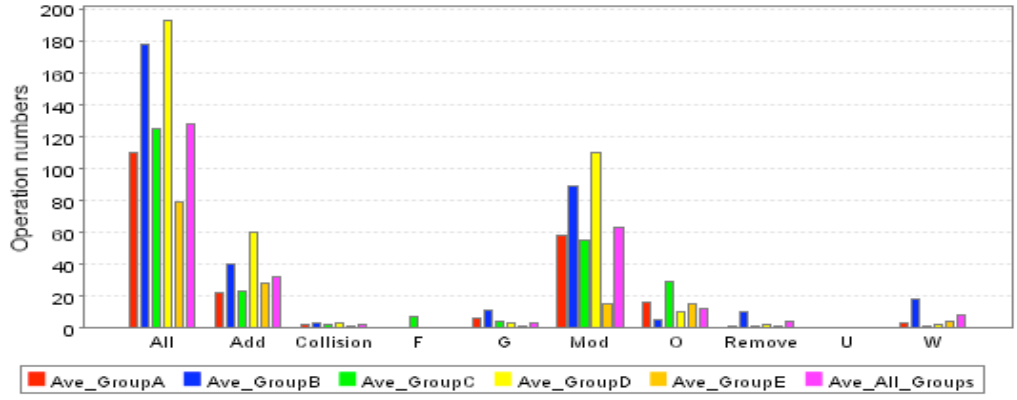

Figure 2: Numbers of operations of all types for the teams

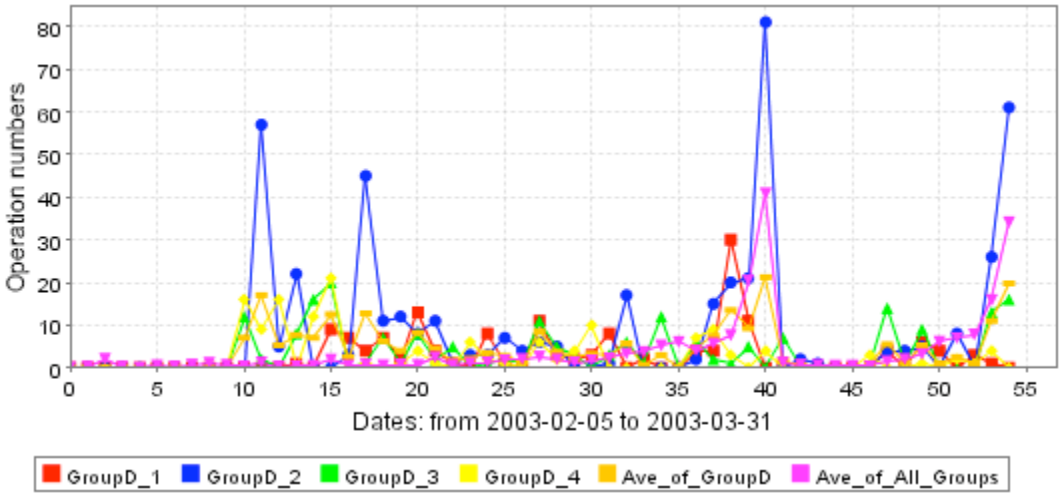

Figure 3: Total numbers of operations of Group D members over time

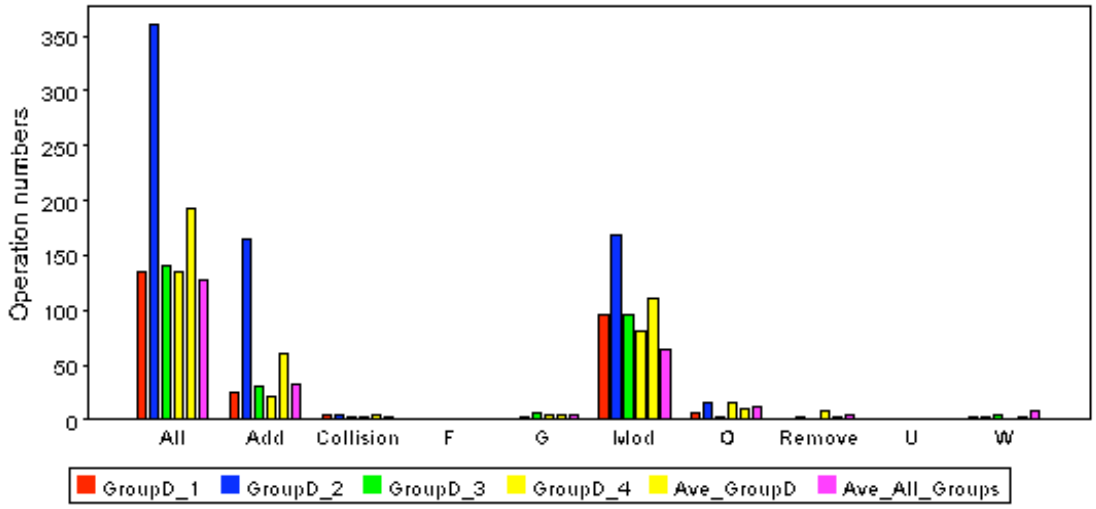

Figure 4: Numbers of operations of all types of Group D members

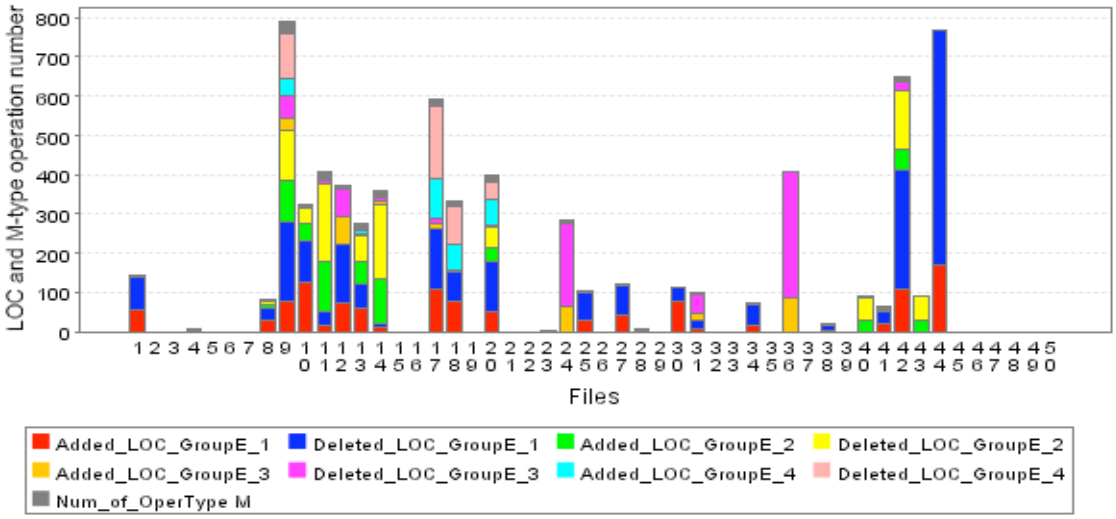

Figure 5: The files of Group E and their revisions

From such diagrams (some omitted here for brevity), we notice that the four members in Groups D and E exhibit different work habits. Group D members started earlier and have a larger proportion of working days. In contrast, all the members in Group E only start work just before the deadlines, with long silent periods according to their CVS repository. Furthermore, Group E had very few revisions per file, suggesting that they may have done most of their work at the very last moment. Student 2 dominated the CVS activity in Group D, with the CVS operations distributed to almost every day of the whole project except the initial planning time. He created many Java files for later completion, and he completed some of them without involving any other members. Although Group D has better habits, their project has an abnormally high number of Java files, with only very few of them being modified. After inspecting the file report, two reasons were found: all the members moved their individual assignment work into the common project directory, and Student 2 dumped another 37 Java files of his own into a directory named "demo/newLayout" on Feb. 22, and never touched them any more.

For group E, although Student 2 started much later, he still ended up adding and changing the most Java files, and took over some Java files from his teammates, being the last to check in most files. Student 3 is an early bird with the least total CVS operations, and the smallest number of Java files related to him, but his work is independent. Student 4 always works hard just before the deadlines, with most of his operations in adding new, complete files. However, his proportion of Java files is very small. These symptoms of poor habits can be correlated to a poor software design as well.

3.4 The students' view

At each deadline, we required each student to submit a self and peer-evaluation for his teammates. Looking at

their responses, we found substantial evidence in support of some of our conjectures in this study.

All four members of Group D had good feelings about their progress. Here are some examples of student comments: "because of my group members work ethics in being determined to start early, work regularly, and keeping each other updated on one another's progress", "communication was open and constant via ICQ and email", "each member was more than willing, if not enthusiastic, to contribute and participate", "I was very impressed with other members' willingness to help other members with problems in their 'assigned' areas". In addition, we got some explicit validation for Student 2's outstanding performance: Student 1 assessed him as "Student 2 did a lot of work with the coding (especially the interface design)", and Student 4 gave such an evaluation: "very impressed with the effort that Student 2 and Student 1 put into the GUI".

Group E also had their own collaboration feedback: most members felt "ok" at beginning although their team had some member changes. However, in their project part 2 report, a lot of problems appeared: "some confusion as to who was doing what. Some parts were done out of order so we couldn't do our part until all this other stuff was built", "some miscommunication of what the plan was", "concentrated largely on the front-end and the back-end was poorly formed and probably will have to be redone for the next part of the project". Student 1 complained about the uneven workloads, and in part 3, he said that some teammates "reverted to the old ways of the computer geek" implying a substantial last minute effort. Finally, Student 4 expressed his feeling that "it is better to underestimate yourself than to overestimate". Changes in the team members was one of the reasons for Student 2's late start; in the end, he had a lot of Java file operations because he "played a key role in trying to integrate the front-end and backend as well as integrating other classes".

From the above students' comments, we can infer that the analyses we discussed here are both valid and potentially effective in noticing problems in a timely way. With the associated simple and easy-to-read visualizations, we are confident that we can help instructors and students to understand their development process better and detect symptoms earlier.

4. Related work

Various CVS repository analysis tools currently exist, such as CVSAnalY [6], CVSMonitor [7], CVSPlot [8], and CVSStat [9]. They collect and present statistical information from CVS files and logs. Most of these tools are mainly suitable for open source projects that span a long period of time. They focus on component size, file revisions, work effort measured in LOC, and all of them can present trends over time with a selectable time granularity. Our research delves more into the types of CVS operations, a wide variety of collected data parameters, and the use of KDD techniques. Ultimately, the goal is to provide a rapid-feedback, process-mentoring tool for novice developers in a small-team environment. The various diagrams we produce are more aimed to assist users in locating poor work habits or work imbalances. Besides having instructors monitoring the teams, the teams themselves can see and reflect on their own progress.

5. Conclusions

In this paper, we described our purpose for analyzing CVS repository histories, outlined the various data parameters recorded in our database, and introduced selected charts and reports for use by instructors and students. Some of the diagrams have been embedded into the Eclipse environment as a plugin. Indeed, an entire environment for process and design mentoring, called JReflex [4] is being developed. Future work consists of polishing this environment, implementing more case studies, and accumulating more data for KDD analysis.

References

[1] D. M. German, *Using software trails to rebuild the evolution of software*, Proceedings of the International Workshop on Evolution of Large-scale Industrial Software Applications (ELISA), September 2003.

[2] J. Han, M. Kamber, Data Mining: Concepts and Techniques, Morgan Kaufmann, August 2000.

[3] Y. Liu, E. Stroulia, *Reverse engineering the process of small novice software teams*, 10th Working Conference on Reverse Engineering, November 13–16, 2003, pp. 102–112, IEEE Press.

[4] K. Wong, W. Blanchet, Y. Liu, C. Schofield, E. Stroulia, Z. Xing, *JRefleX: towards supporting small student software teams*, OOPSLA Workshop on Eclipse Technology eXchange 2003, pp. 50–54.

[5] CVS, http://www.cvshome.org

[6] CVSAnalY, http://barba.dat.escet.urjc.es/index.php?menu=Tools&Tools=CVSAnalY

[7] CVSMonitor, http://ali.as/devel/cvsmonitor/

[8] CVSPlot, http://sourceforge.net/projects/cvsplot

[9] CVSStat, http://www.gnu.org/directory/all/cvsstat.html