SubCollaboration: large-scale group management in collaborative learning

Abelardo Pardo*, and Carlos Delgado Kloos

Department of Telematic Engineering, University Carlos III of Madrid, Spain

SUMMARY

Computer-supported collaborative learning is a paradigm that uses technology to support collaborative methods of instruction. When combining collaborative learning with the need to exchange documents between students and the teaching staff in a blended learning scenario, version control systems (VCSs) greatly simplify this collaboration. Furthermore, these tools need to be adopted in regular classes as they are used in industrial environments. But deploying a collaborative environment in which version control is used does not scale for large classes. This paper presents SubCollaboration, a platform that uses the VCS Subversion to manage a large number of work spaces in a collaborative learning environment. The tool maintains a reference workspace where teaching staff introduces new material that is then synchronized with the team repositories. Two case studies are presented showing that students easily learn the use of version control and its deployment in large classes is feasible.

KEY WORDS: version control systems; computer-supported collaborative systems; software management

1. INTRODUCTION

Team collaboration for software development is present in multiple engineering degree programs. As recent studies suggest [1], team projects are important components of undergraduate curricula to expose students to professional practices. This same tendency can also be observed in the general context of the Internet where, in the recent years, numerous web-based portals for software projects have appeared (see [2] for a survey). Version control systems (henceforth, VCSs) are tools used in these scenarios to exchange documents among team members and maintain the history of changes to recover any previous version if needed. There are numerous VCS tools available and their functionality has been increasing steadily over the years.

In educational institutions, however, these systems are typically introduced (if at all) in the last courses of a program. But as stated by Reid and Wilson [3], there is clearly a pedagogic value in using them also in the early courses. Other authors even consider the use of these tools 'imperative' [4]. But the effective use of VCS in a collaborative learning environment faces two issues. First, students need to learn the use of these systems in advance. If the tool is used to support the regular activities of a course, this effort needs to be kept to a minimum. The second problem is to scale the use of shared workspaces to large classes. In a typical collaborative environment documents are exchanged not only among teammates, but also among students and the teaching staff. Furthermore, the teaching staff may require to gradually add material or change documents

*Correspondence to: Abelardo Pardo, Universidad Carlos III de Madrid, Av. Universidad 30, 28911 Leganés, Madrid, Spain.
E-mail: abelardo.pardo@uc3m.es

for the workspaces of all the teams. These two issues may render using VCSs unfeasible for scenarios with a large number of students.

The paper presents SubCollaboration, a tool that using the functionality offered by Subversion [5] maintains a reference workspace synchronized with the workspaces created for the teams derived from the organization of a set of users. The tool creates one workspace per team, sets the appropriate permissions, and merges the changes appearing in the reference workspace. This reference workspace is where the teaching staff first creates and then maintains the basic structure of all team workspaces. The tool provides a simple interface to manage administrative tasks such as team creation, deletion, and modification that can be very time consuming when the number of teams is large.

This work has two objectives. First, deploy the use of VCS in a collaborative environment because these tools are frequently used in industrial scenarios. Second, provide a tool to support the management of a large number of team workspaces synchronized with a reference workspace. This tool must have the following features:

- Automate the tasks derived from group management. Perform, whenever possible, one single operation in a repository that is then automatically replicated for all other repositories.
- A role-based fine grain policy to access repositories. Allow teaching staff to access all resources, students to access their own folder, maintain read-only folders in each student workspace, and allow folders to change from read/write to read-only over a time window.
- Offer an environment to monitor team progress. Provide customized views of the workspaces for the teaching staff (for example, the student workspaces of the teams in a class section).

The remainder of the document is organized as follows: Section 2 reviews the related work and the latest contributions in the area. Section 3 describes the tasks derived from a collaborative scenario with a large number of teams. Section 4 describes the two users' views, students and teaching staff. The main algorithms used in the design are described in Section 5. The experimental procedure to validate the tool is explained in Section 6. Section 7 shows the results obtained in the two case studies. Finally, Section 8 presents the conclusions as well as avenues for future research.

2. STATE OF THE ART

The work described in this document is influenced by contributions in the fields of computer-supported collaborative learning (CSCL) and Technology Enhanced Learning.

CSCL was presented by Koshmann in 1996 [6] as a new paradigm that uses technology as a mediational tool to support collaborative methods of instruction. CSCL has evolved from the early projects providing a new medium for textual communication or collaborative text production, to a variety of research groups and a large body of literature [7].

The space of collaborative learning contains a large number of possible scenarios that can be categorized along three dimensions: group size and time span, the type of learning, and the type of collaboration [8]. The tool described in this paper targets the interaction of a very large number of small to mid-size groups over a potentially large time span (from weeks to a year), jointly involved in solving a problem requiring the exchange of documents supervised by a set of tutors, and with a set of objectives that need to be collaboratively achieved.

The use of version control platforms in educational environments has been considered by multiple authors. However, the proposed methods cannot be applied when deploying these tools in the context of a collaborative learning scenarios with a large number of users. In [9], Lee *et al.* present a tool for collaborative writing. They discuss the need to control the access to different parts of a document based on roles. These requirements were not met by conventional version control tools; hence, a specific solution was created. Although the underlying model was collaborative, there was no mention as how to cope if the environment had to be replicated for a large number of communities each writing a document.

Numerous administrative tasks can be automated with the use of a version control system. In [10], a module is described that when embedded in an integrated development environment, students may submit their work to a central server where some tests are executed. The results of these tests are sent back to the students. This same technique can also be extended to facilitate the exchange of arbitrary feedback, auxiliary documents, etc. In this case, the level of automation is located within each group, but no mention is given about how to manage multiple groups with the same scheme.

Reid and Wilson [3] provide a detailed description of the use of CVS to manage student submissions and expose them to this type of a tool. They point out that several authors consider the use of a VCS as a factor to contribute to the success of a project. But as the authors acknowledge, CVS was not designed for educational environments, and therefore, issues such as fine grain access control, accepting late submissions, or the administrative overhead when students move between teams are unsolved.

In [11], Hartness presents an approach to combine the integrated development environment Eclipse with the version control tool CVS [12] for group projects. The advantages of having this tool for teamwork is explained, but issues such as workspace access policy, group member management, or how feedback is produced are not discussed. The paper also mentions the possibility of analyzing the information collected by the version control server. Every time a new version is committed to the repository, date, time, user, and a comment are also recorded. The use of this information to monitor user activity has also been the center of additional work [3, 13, 14], but the quality of the extracted information has been less than expected.

In the work presented by Glassy [15], Subversion is used in a senior-level course of 11 students to reveal the software development process in a time-efficient way. The approach required students to create their repositories from scratch, thus dealing with sophisticated administrative tasks to initialize a repository and manage the access permissions. Although passing these tasks to the students clearly helps in managing large-scale courses, it is not suitable for freshman or sophomore-level courses. Besides, if the repositories are created incorrectly, the administrative effort by the teaching staff to solve these anomalies may be potentially very high. Furthermore, the exchange of documents between students and staff was still done through e-mail, because the student repositories were not centralized.

The tool presented in this paper is similar in spirit to SVNLecture [16]. The teaching staff interacts with the tool through a web interface, and a client program is provided for students. But the web interface is not adequate when the number of groups to manage is large. Furthermore, there are several graphical interfaces available, thus rendering the specific client application unnecessary. SubCollaboration can be considered as an evolution of this tool because it abstracts even further the group management tasks to make them suitable for large-scale classes. The teaching staff operates directly on a list of group members, and the operations to create access permissions, clone repositories, or modify team configurations are all derived automatically for all the groups.

The problem of large-scale management and the lack of adequate mechanisms in current CMSs was described in [17]. CMSs tend to be *browser centric*; as a consequence, tasks such as distributing submitted material among different lab sections, access to all submitted material efficiently, or returning personalized feedback are very time consuming. The proposed solution is based on a combination of several commercial off-the-shelf components. Using a VCS would have provided the ideal setting to accomplish most of the tasks mentioned in this paper.

An example of the need for version control tools to maintain the connection between team workspaces and a reference workspace controlled by the teaching staff is described [18]. Students are given a virtual appliance and use version control (git [19]) to obtain a copy of the source code for the Linux Kernel with more than 10 million lines of code. The teaching staff may introduce changes in the Kernel that need to be propagated to all the team workspaces. Access control relies in the Unix file system functionality, and therefore, has a coarse granularity.

Another example of the advantages of using version control in a collaborative setting is presented in [20]. Although the system is oriented only to computer science courses, Helmick points out the advantage of using version control so that the teaching staff can easily review the progress of the students. The proposed approach uses Subversion, but is based on a per-student repository, and collaborative scenarios are not considered. This approach makes the administration of a large number of teams unfeasible.

In [21] Meneely and Williams describe Jazz, a synchronous collaborative development platform and its use by software engineering students [21]. The emphasis of the system is on tool integration. Instead of using multiple programs to obtain a software development environment (Eclipse, Subversion, Bugzilla), Jazz offers all this functionality in a single environment. Students need to learn how to use this environment, and therefore it is proposed for third and fourth year courses. The tool proposed in this document offers a much simpler interaction with the repository and therefore does not require any prior knowledge about software development environments.

3. THE LARGE-SCALE GROUP MANAGEMENT SCENARIO

The work in this paper is presented in the context of the following scenario. The second-year course has a total of 200 students who are divided into five sections. In each section, students are organized into teams of four or five to work collaboratively in a project. For the sake of the example, a total of 45 teams is assumed. All 45 teams work in the same project but independent of each other. Each team is assigned a tutor, and each tutor supervises no more than 10 teams with a total of 5 tutors. The project is described in a document containing a description of the objectives, the requirements, and a clearly stated set of milestones and deliverables.

The set of milestones and deliverables requires a continuous exchange of material between the team members and the tutor. At different points during the project development, the teaching staff makes certain documents available to the team. Analogously, at certain stages, the students send the deliverables to the tutor. Aside from these formal document exchanges, the teams are free to create as many additional documents as necessary in their workspace.

All students participating in the project go through a sign-up step from which a set of teams are derived. SubCollaboration does not give any support for this step. It simply takes a file stating how the students are grouped (see Section 4 for a detailed description).

Simultaneous to the team creation, the teaching staff creates an arbitrarily complex structure of folders and subfolders containing the set of initial documents. This repository will be called the ‘reference workspace’. A subset of these folders can be defined as ‘read-only’ and as such, students cannot modify their content. Although the content of these folders could be available through the web, they are included to make sure that the workspace is self-contained even when no network connection is available. The rest of them are allowed to be modified by the students. Also, a so-called ‘active window’ can be defined for any folder in this workspace by means of two time stamps. The first is the time after which the folder becomes available to the students. This is to allow the teaching staff to work in portions of the workspace that are not yet visible to the students but will be in the future. The second time stamp marks the deadline after which no changes are allowed in the folder. The use of this time is to enforce submission deadlines. Thus, a folder with these two time stamps is initially only available to the teaching staff; when the first time stamp is reached the permissions are changed to allow students read and write operations, and when the second time stamp is reached, the permissions are again changed to ‘read-only’.

These changes in permissions are achieved using the path-based authorization feature in Subversion with the execution of SubCollaboration commands at regular intervals. Subversion allows to define rules stating the level of access for a user to access any directory. SubCollaboration transforms the timestamps of an active window into Subversion access rules. The change in permissions requires re-processing this file. Using an application to execute scheduled commands, the rules are refreshed at these times specified by the active windows. A large number of workspaces or directories with active windows produce a sizable set of rules that could not be handled manually.

Once an initial version of the reference workspace has been created, SubCollaboration creates as many copies as the number of defined teams. Thus, each team is given access to their initial repository with an exact copy of the reference workspace. This repository is obtained and managed using regular subversion commands. All team workspaces are managed together

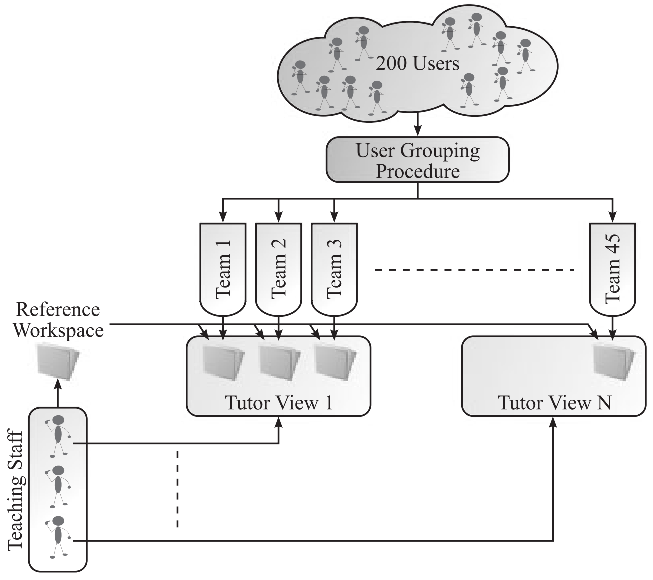

Figure 1. Sample scenario to use with SubCollaboration.

in the same Subversion repository. For each tutor, a special view is created in which only the workspaces of the teams being supervised are included. Still, to simplify administrative tasks, all members of the teaching staff are given permission to access (both read and write) all workspaces.

Throughout the duration of the project, the teaching staff may modify the reference workspace by either modifying documents already available to the students, or adding new folders with additional documents. Simultaneously, students work in the project and submit changes and obtain the latest version from the VCS regularly. An illustration of the proposed scenario is depicted in Figure 1.

This scenario is conceived such that it can be scaled along the following dimensions: overall number of students, number of students per group, number of groups per tutor, and number of tutors. The intuitive administrative tasks to handle the structure depicted in Figure 1 (new team, new file in the repository, change a deadline, permissions on a folder, etc.) can be done manually in most VCS tools for few teams and tutors. But, when the number of teams is large, as in the described scenario, the time cost of these tasks rules them unfeasible. SubCollaboration contains the scripts to replicate these tasks from a given set of groups and simple configuration parameters.

The first step when deploying this scenario is group creation. Our experience shows that due to the high number of possible strategies, including this functionality in the tool would restrict its possibilities. For this reason, groups are assumed to be created in advance and stored in a plain text file containing a space separated list of user identifiers per line. The choice of this simple format has several advantages. The file can be created using a large variety of tools. For example, from a spreadsheet used by the teaching staff to maintain scores and annotations, a new sheet can be created and exported in a CSV format (using spaces as separators). This aspect is important in large classes when group creation can be done separately by several members of the teaching staff. Also, this format allows the file to be quickly inspected to make small adjustments (as described later). Figure 2 shows an excerpt of this file with the description of five teams with 9-digit user identifiers (content beyond '#' is ignored).

SubCollaboration has been conceived to work as part of a learning environment with certain technological support. It assumes a community of users within an institution that can be authenticated with a user identifier. Currently, Lightweight Directory Access Protocol (LDAP) is supported. Aside from the file containing the group members, SubCollaboration uses an additional set of

```
#
# Section 5 (5 teams)
#
100066879 100072751 100060691 # Team 38
100066981 100081433 100081435 100066983 # Team 39
100081436 100073284 100074810 100081397 # Team 40
100073325 100081398 100066842 100081472 # Team 41
100081407 100068925 100077072 100081483 # Team 42
```

Figure 2. Sample of the group configuration file.

configuration variables containing administrative information such as:

- Path to the local subversion repository.
- Path to the reference workspace.
- Prefix to use when creating the groups.
- List of user identifiers for the staff members.
- List of read-only directories for the groups in the reference workspace.
- Active window restrictions for folders.
- Connection parameters for LDAP authentication.

The tool is invoked with a command line interface with the following format:

```
sc [command] [options]
```

The first step is to take a folder already under version control by Subversion and initialize the required data for SubCollaboration. This tasks is simply done with the '`sc init`' command. The initialization process assigns default values to all the required administrative variables. These values can be viewed and changed from the command line with '`config get`' and '`config set`' commands. For example, the variable '`users.staff`' contains the space separated list of user identifiers of the teaching staff. The value can be set with the command:

```
sc config set users.staff john susan bob janet mike lisa
```

Staff needs to be distinguished from the rest of the users because they are granted read/write permission for all groups. The next step is to create the initial groups from a text file with the format previously explained.

3.1. Group operations

After setting the proper values in the administrative variables, the next step is to create the group descriptions. Given a plain text file with the format previously described, the groups are created with the following command:

```
sc group create -i -f Groups.txt
```

This command creates the groups as specified in the file. Option '`-i`' instructs the tool to work incrementally with the group structure already defined in the folder (if any). This is useful when groups are added to the current set, as explained later. The operations offered to manipulate group configurations are the following:

Add/delete/move a user. Add, delete, or move a team member.

Create a new group. Add a new group with the given set of user identifiers.

Fetch LDAP information. Obtain user name and e-mail of each group member (only if the appropriate administrative variables are configured).

Notify group members. Given a template file and a set of groups, create a script that sends an e-mail with the template modified with the group name to each group member. This command is used to notify users about how to access their workspace.

Lock/Unlock group workspace. Turns on/off access permissions of the group workspace in the repository to the group members. This command is used by the teaching staff to review the group documents without interference.

Show group information. Shows the name and e-mail of all members of a given group.

Search groups. Regular expression search over the group info.

User group. Shows the group information about a given user

These commands aim at automating regular group management tasks that are time-consuming when applied to large classes. Group notification is an example. Once the groups are created, users need to be notified of their group name and how to access the working space, the name of which depends on the group name. With this command, all groups are notified instantaneously and therefore can start to work with their documents.

Locking and unlocking a set of workspaces is another useful command. For example, let us assume that teams are working on a project for which there is a partial version that is to be reviewed by the teaching staff. The administrator may lock the directories briefly to allow the staff to copy the appropriate documents to review. This amounts to changing of all the permissions for all the groups in a Subversion configuration file. SubCollaboration simplifies this task with a single command.

The search capability over group information and user names is provided to optimize the communication flow between students and staff. Our experience shows that the numerous questions raised by the students are sent either through e-mail or a post in the course forum. When responding to these questions, the teaching staff needs to quickly locate a group by the name of one of its members.

3.2. Replication of the reference workspace

Once the groups are created, the next step is to replicate the content of the so-called 'reference workspace' once per group. This is the most time-consuming task for SubCollaboration. For each team, the proper Subversion commands are issued to replicate the same structure of documents and folders, but as a different folder inside the repository. This operation is invoked with the '`align`' command as it performs an alignment between the group and the reference workspaces. The resulting set of folders within the repository is illustrated in Figure 3. By maintaining all these folders in the same Subversion repository, the access to the proper workspaces is simplified for the teaching staff (one single URL is used to checkout the appropriate workspaces).

From this point on, these spaces will evolve independently over time. The group workspace is modified by team members, whereas the reference workspace can be modified by the teaching staff. However, SubCollaboration must maintain the relation between the group and reference documents to be able to propagate changes. This situation arises when, after aligning all workspaces with the reference, a change is produced in one of the reference documents. Because the workspaces are independent, these changes are not propagated to the groups with conventional VCS commands.

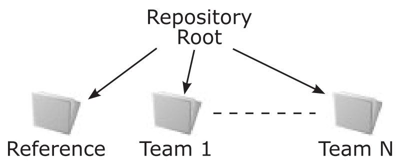

Figure 3. Top-level folders in the repository after replication.

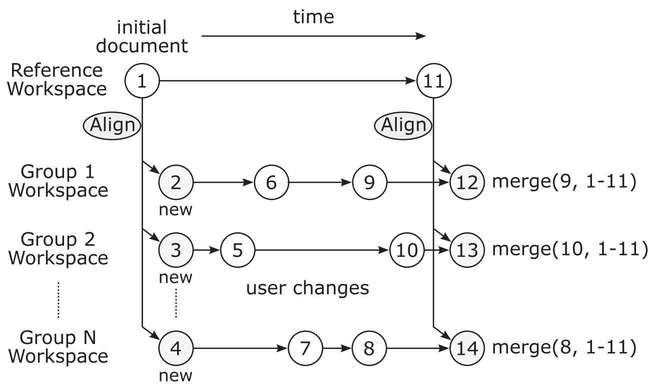

Figure 4. Merging changes from the reference workspace.

The '`align`' command takes into account this situation. When an alignment is performed, the program first checks whether the files in the group workspace need to be created for the first time, or updated with respect to some previous version. In the second case, the file in the group workspace cloned from the reference could have been modified by the users, and therefore, a 'merge' operation is required. Figure 4 illustrates this situation.

The merge operation is different for each group workspace. In the example in Figure 4, the changes stored in the reference workspace from version 1 to 11 need to be merged with version 9 for Group 1, version 10 for Group 2, and version 8 for Group N. As it is the case with any merge operation in version control platforms, this operation has several restrictions. For plain text files, the version control program tries to isolate the area where changes are applied, and if they are disjoint, the merge can be performed without user intervention.

However, if the changes to merge are in areas already modified, the user intervention is required. In this case, Subversion surrounds the area of the change and offers the two possibilities for the user to choose. In the current scenario where SubCollaboration is applied, the potential for a large number of conflicts and, therefore, of user intervention is present. However, as shown in Section 7, during our experiments, although documents in the reference workspace changed frequently, user intervention to solve merge issues was seldom required.

3.3. Permission management

Managing in the same central repository the reference workspace, one workspace per group, and the additional administrative files requires a fine grain permission system. Subversion offers a powerful permission mechanism where access can be allowed in a per directory, per user basis. Any attempt to operate over the repository is checked against an authorization file.

Creating this authorization file manually is another task that cannot be performed for large-scale classes. The policy implemented by SubCollaboration is based on the following premises:

- An authorization group is defined for the teaching staff.
- An authorization group is defined for each working group.
- The teaching staff group is given read/write access to the entire repository.
- Each group is given read/write access to the root of its group workspace.
- For each group, access is restricted to 'read-only' to those directories defined as such in the configuration file.
- For each group, assign the appropriate permissions for the folders with active window restrictions.

The '`auth`' command in SubCollaboration creates the authorization file following the Subversion syntax and copies it in the appropriate location in the repository. Folders with 'read-only' permissions are conceived to provide auxiliary documents to each group that cannot be modified by group members. An example of the use of these folders is to offer auxiliary code libraries, or a folder where the staff writes the review of the submitted work. The staff has full access to all folders, and therefore may deposit the reviews that will appear in the local copies of the group members after a regular update operation.

4. SUBCOLLABORATION: USER VIEW

SubCollaboration assumes two user profiles: a set of users divided into groups who are allowed to operate a folder or workspace under the control of Subversion, and a set of users in charge of maintaining the reference workspace, which is gradually populated with documents and made available to the rest of the users during specific time windows. In the educational scenario used in this document (see Section 3), the first type of users are the students, and the second the teaching staff.

4.1. Student view

Students start the interaction with SubCollaboration receiving an e-mail in which they are notified of the URL to access their workspaces. This message has been created by the teaching staff with a SubCollaboration command after group definition.

After a regular Subversion ‘`checkout`’ command, each student team member has access to a copy of the initial version of their workspace. This area has three types of folders depending on their access rights. Certain folders are ‘read-only’ and no file changes are allowed. New files can be created in these folders, but they cannot be uploaded to the central server, thus remaining in the user local copy. Regular work is carried out in folders with read/write permissions. Students may add new documents or subfolders freely in these folders. A third set of folders with no access permission, which might be already present in the user workspace, are not downloaded from the Subversion server. Figure 5 depicts an example workspace with one folder of each type.

The read-only folder contains templates for the source files. Although these files could be available in other locations, their inclusion facilitates the operation in the repository when not connected. The folder with read/write permissions contains the files created by the team during the project. The folder with no-access contains a midterm version of the project that is currently being reviewed by the teaching staff. The folder is still visible to the users, but no changes can be uploaded or downloaded from the repository. Upon termination of the review, the folder permissions can be turned to read-only and changes can be downloaded from the server.

4.2. Teaching staff view

The teaching staff duties derived from the use of SubCollaboration are kept to a minimum. An administrator (he could perfectly be a member of the teaching staff) initializes the repository and defines the required variables, mainly, the list of staff users, directories with read-only access, and the time stamps for the directories with an active window. These definitions can be done after or

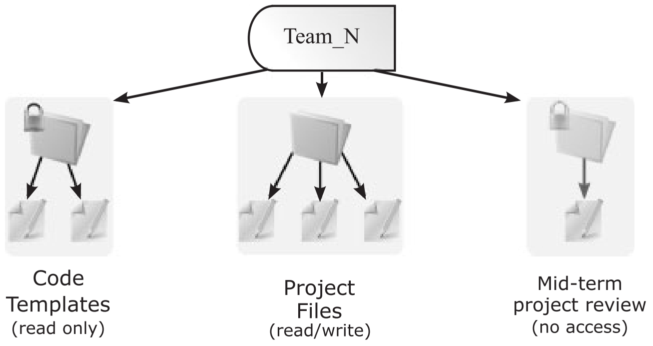
Team_N
Code Templates (read only)
Project Files (read/write)
Mid-term project review (no access)

Figure 5. Student workspace with three types of folders.

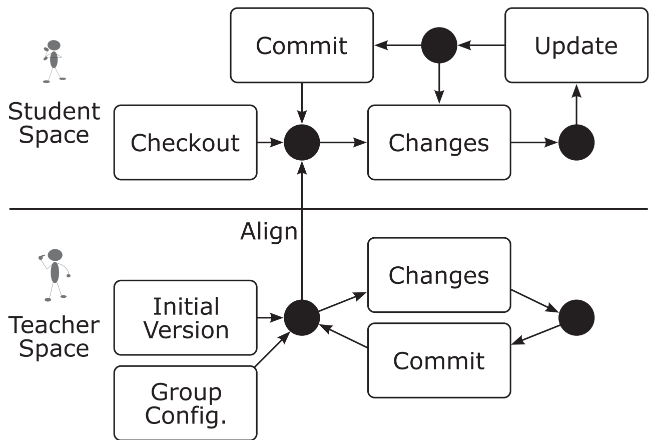

Figure 6. Activity diagram of the two user views.

during the creation of the reference workspace. Groups are then defined and uploaded to SubCollaboration. The administrator then generates a script from a given template with the commands to send the notification e-mail informing students about the location of the assigned repository.

Once an initial version of the reference workspace is ready, the administrator proceeds to clone the workspace for each of the defined teams with the proper permissions. This step includes the generation of a new authorization file that is copied in the Subversion repository. The reference workspace may contain the three folder types previously described: Read-only folders with auxiliary documents, read-write folders with the material to be managed by the students, and no-permission folders with material that is not yet ready to be seen by the students. With this scheme, teaching staff may work in the reference workspace at the same time as the teams work in their respective copies. From this point on, any time a member of the teaching staff changes the reference workspace, the change needs to be propagated to the team copies as described in Section 3.2. Figure 6 shows the activity diagram combining the student and teacher view.

The changes in a team configuration are also easily managed by SubCollaboration. Groups can be added, modified, or removed at any point in time. The administrator reflects these changes in the configuration file and a new permission file for the entire repository is generated. When a student is removed from a group, no additional operations are allowed from her repository. If a new student is added to a group, a script is generated again with the same template with the command to send an e-mail containing the location of the new working area. Changes in permissions for the directories are also easily implemented. Whenever there is a change in the permission policy, the authorization file in the repository is updated with immediate effect.

Aside from making the content gradually available to students, the active window functionality for folders provided by SubCollaboration can also be used to provide personalized resources to the teams. Let us assume that each member of the teaching staff has to review and provide feedback about the work of the teams in a section (see Section 3). This feedback will be included in a folder created for that purpose in the reference workspace. The administrator then defines this folder as ‘no-permission’ and performs a clone operation. The new folder is now in each team workspace. Each member of the teaching staff then adds the appropriate documents for each team in that folder. Once all newly created folders contain the personalized resources, the administrator changes the permissions (either to read–write if students are allowed to record changes in these resources, or read-only) and the folders are available to the teams in the next update operation.

5. DESIGN AND IMPLEMENTATION

SubCollaboration has been designed to take advantage of the functionality already available in Subversion and enhance it to manage a large number of groups. The defined data structures

```
METHOD CreateAuthorizationFile( config )
BEGIN
    reposirotyRoot ← config.getOption('svn.root')
    staffGroup ← DefineGroup( config.getOption('users.staff') )
    GrantPermission( staffGroup, repositoryRoot, 'rw' )
    FOREACH group IN config.getGroups() {
        gr ← DefineGroup( group )
        FOREACH restrictedFolder IN config.getRestrictedFolders() {
            permission ← 'rw'
            IF now() < restrictedFolder.getSinceTimeStamp() {
                permission ← ''
            } ELSEIF now() > restrictedFolder.getUntilTimeStamp() {
                permission ← 'r'
            }
            GrantPermission(gr, restrictedFolder, permission)
        }
    }
    allGroups ← DefineGroup( config.getGroups() )
    FOREACH readonlyFolder IN config.getReadOnlyFolders() {
        GrantPermission(allGroups, readonlyFolder, 'r')
    }
    WritePermissionFile()
END
```

Figure 7. Algorithm to generate the Subversion authorization file.

are all contained in two configuration files: one containing the group definitions and a second file with the remaining variable definitions. The configuration operations (set/unset variables, create/delete/modify groups, etc.) are directly reflected in the configuration files. The format of these files is inspired by the INI format[‡] to facilitate visual inspection with an editor. All the operations implemented by SubCollaboration modify the configuration files (create new team, modify team members, etc.), modify the Subversion server configuration files, or translate into a set of Subversion commands to apply to the repository (for example, when aligning the reference workspace with the team workspaces).

One of the most resource-intensive operations in SubCollaboration is the creation of the authorization file. The files with the team members and folder definitions are read, and the Subversion authorization file is generated with the algorithm illustrated in Figure 7.

The operations *DefineGroup* and *GrantPermission* write the appropriate rules with Subversion syntax in the authorization file. For a large number of groups, the size of the resulting file clearly shows the need for automatic processing. The presence of restricted folders (those with an active window) require this algorithm to be executed at regular time intervals for the permission to work as expected. No basic Subversion commands are generated.

The other important operation in SubCollaboration is the propagation of the reference workspace to the team repositories. The reference workspace is compared with the existing team repositories and the appropriate basic Subversion operations are scheduled and executed in the repository. The code for this recursive algorithm is shown in Figure 8.

The procedure first creates the destination workspace if needed. Each file and folder in the source directory is traversed. For folders, a recursive call is executed with the analogous folder in the destination. For a file, there are two possible cases. If the file in the source workspace does not exist in the destination, it is created and added to the repository (Subversion 'add' operation is scheduled). If the file also exists in the destination workspace, a Subversion 'merge' operation is scheduled. All changes in the source file from the execution of the last alignment need to be merged in the destination file. The version of the source workspace in the last align operation

[‡]http://msdn.microsoft.com/en-us/library/bb158614.aspx.

```
METHOD AlignWorkspaces ( source, destination )
BEGIN
    IF not destination.exists() {
        CreateDirectory(destionation)
    }
    FOREACH file IN source.getFiles() {
        IF file.isDirectory() {
            AlignWorkspaces ( file, destination/file )
        } ELSE {
            IF not destination/file.exists() {
                CopyAndAddToSubversion(file, destination/file)
            } ELSE {
                currentVersion = GetCurrentVersion(source)
                lastMerged = GetLastMergedVersion(destination)
                Merge(source, lastMerged, source, currentVersion, destination)
            }
        }
    }
    CopySVNProperties(source, destination)
END
```

Figure 8. Algorithm to align the reference workspace with the group workspaces.

is stored in a property in each destination file by executing the ‘propset’ Subversion operation. Furthermore, if Subversion properties have been defined for a file, they are also propagated to the destination workspace with the ‘propset’ Subversion operation (the set of properties to clone is configurable).

If the merge operation is not successful (that is, changes could not be automatically combined), the file is marked as in conflict and the administrator is notified to finish the merge as it is conventionally done in version control platforms. The algorithm concludes by copying Subversion properties to the destination folder.

Once all the basic subversion operations have been executed, the changes are committed to the repository. At this point, all workspaces are considered to be ‘aligned’.

6. THE EXPERIMENTAL PROCEDURE

The two premises for the creation of the presented tool were to introduce students to the use of version control in a collaborative environment and to manage these workspaces efficiently to make its use feasible in large classes. Several studies point to the correlation between student satisfaction and achievement, and others establish also a relationship with educational quality [22]. It is for this reason that student and faculty satisfaction were taken as figures of merit to measure the level of engagement.

The second premise of efficient management is measured by comparing the percentage of work that has been automated. The figure of merit is the number of basic operations over the repository required during a semester. If the number is sufficiently large, the tool clearly offers support for an otherwise unfeasible environment.

6.1. Hypothesis

Considering the premises previously described, the proposed study poses the following hypothesis:

Hypothesis. Students will learn to use a VCS without a significant effort in a collaborative environment and perceive it as a beneficial skill toward their professional life.

6.2. Methodology

The methodology used for the study is based on two instruments. The first is a survey given to both students and faculty about the impact of the use of SubCollaboration during the course. Students were asked about how useful was the use of VCS, the type of use (exchange files among peers, with the teaching staff, among different computers), the effort required to learn the software, etc. The teaching staff was asked the same questions and additionally how important they considered the use of version control by the students.

The second instrument used is a detailed log of both the operations, basic Subversion and SubCollaboration operations. Every SubCollaboration command executed by the staff member was recorded as well as the number of basic operations over the Subversion repository. Handling multiple working spaces will clearly have a multiplicative effect, but in order to verify the hypothesis this effect needed to be carefully assessed.

6.3. Case studies

The paper reports two case studies carried out both in the fall semesters of years 2008 and 2009.

Case 1: *Operating systems*

The first case study is a fourth year undergraduate course on operating systems of the core degree of Telecommunication Engineering. The course topics include process scheduling, memory management, input/output operations, and file systems. It covers units CE-OPS3, CE-OPS4, and CEOPS-7 of the Computer Engineering Curricula 2004 proposed in [23]. The course has five C programming assignments related to the topics covered in the theory classes. Students register to work in pairs. Each assignment has from 1 to 2 weeks of out-of-class work and starts with a 2-h lab session in which the tutor explains the assignment and supervises the work.

The study was carried out from September to December 2008 with 84 students, 48 groups, and 5 teaching staff members. Students registered in pairs at the beginning of the course and received an e-mail notifying them of the URL to check out their working space. A document was made available at the beginning of the course explaining the basic Subversion operations.

This case study was mainly used to prove the first hypothesis. A survey was conducted at the end of the course to evaluate the level of engagement of students with the system and the usefulness perceived by the teaching staff.

Case 2: *C Programming*

The second case study was done in the second year undergraduate course on C programming of the core degree of Telematic Engineering. Course topics include C data structures, pointers, and dynamic memory management. It covers units CE-PRF2, CE-PRF3, and CE-PR4 of the Computer Engineering Curricula 2004 proposed in [23]. The study was carried out from September to December of 2009, had a lab session per week, and lasted for 14 weeks. The experience was divided into two phases. In the first half of the course, students worked in pairs in a scheme similar to the previous case study. During this phase, a total of 208 students organized into 109 groups, and four teaching staff members participated. In each week a new folder was created in the reference workspace and propagated to the student workspace with the material required for the laboratory session.

During the second half of the semester, teams of four students were created by the teaching staff to work on a single project. A description including deliverables, milestones, and evaluation criteria was published at the beginning of this period. A workspace was created for each group containing some initial documents and folders. A total of 152 students divided among 48 teams and four teaching staff members participated. In both phases of this case study, SubCollaboration was used to manage the workspaces.

Owing to the large number of participating students, this case study was used to prove the second hypothesis. All operations carried out by SubCollaboration were then logged to calculate the number of basic Subversion operations required.

7. RESULTS

Case 1 was used to analyze how students used version control within the proposed collaborative environment. A survey with 10 questions was published during the last week of the course. Each question had to be answered on a scale from 1 (totally disagree/almost never) to 5 (totally agree/very frequently). A total of 37 answers were received. The obtained results are shown in Table I.

The answers for all questions, except for question 5, are close to the extreme values with reasonable standard deviation. The first question shows that students had barely any previous experience with the version control software. Questions 2–4 show an equally intensive use of the system to exchange files among teammates, with the teaching staff and with different computers. The results shown in question five leave the issue of the appearance of conflicts inconclusive. These events are produced in the client, and therefore no trace of them is left on the server.

Question 6 shows how students acquired the habit of committing their files after a work session. The results obtained for question 7 are perhaps the most significant. An average of 3.91 and a deviation of 1.17 confirm that the effort by the students to learn the tool was negligible. The distribution of answers for the five possible levels (level 1: totally disagree to level 5: totally agree) was, respectively, 1, 5, 4, 11, and 14. Only six students (17.2% of the total) seem to require extra effort to learn the use of Subversion.

Question 8 proves another relevant point. The tool is perceived as useful for the future and therefore student engagement can also be considered high. Question 9 confirms that most of the interaction with the VCS was done with the basic commands. Finally, the last question shows that students perceived the potential for this type of file management to be useful beyond the current course. Additionally to these data, the repository registered a total of 1554 commit operations by the students, an average of 18.5 operations per student.

From these results, it is clear that students did learn how to use the VCS with almost no effort (Question 7), they exchanged files with teammates, staff, and themselves (Questions 2–4), and they perceived it as a positive skill to learn (Question 8).

The survey was also answered by the five members of the teaching staff with the results shown in Table II.

The results are consistent with those obtained from the student survey. The proposed environment was useful to manage the student work, only a basic set of commands were required, and no extra effort was needed to manage the files. Also consistent with the student view, conflicts during merge operations were not perceived as an important issue.

But as previously described, proving that the use of version control in a collaborative environment is positive to students is only one part of the problem. The complexity of managing these workspaces concurrently may require an extra effort from the teaching staff that renders the approach unfeasible. Case 2 was used to show the effect of managing this environment with SubCollaboration from the point of view of operations in the repository.

Table I. Student survey results.

No.	Question	Mean	STD
1	I used version control tools before this course	1.69	1.21
2	I used Subversion to exchange files with my teammate	4.22	1.02
3	I used Subversion to exchange files with the teaching staff	4.72	0.57
4	I used Subversion to use the same files in different computers	4.67	0.79
5	Conflicts between different file versions appeared	2.92	1.44
6	I submitted my changes upon finishing my work session	4.03	1.32
7	Learning Subversion required no extra effort	3.91	1.17
8	Knowing these tools will be useful in my professional career	4.20	0.76
9	I have used more commands aside from the basic ones	1.89	0.98
10	This system may be useful for non-programming courses	4.19	1.06

Table II. Teaching staff survey results.

No.	Question	Mean	STD
1	Version control was useful to review student code	4.6	0.80
2	Version control was useful to see the activity for each lab	3.8	0.75
3	Conflicts between different file versions appeared	2.4	0.80
4	The use of Subversion required no extra effort	4.4	0.49
5	I have used more commands aside from the basic ones	1.4	0.49
6	This system may be useful for non-programming courses	3.2	0.75
7	The use of this tool is needed by the students	4.2	0.75

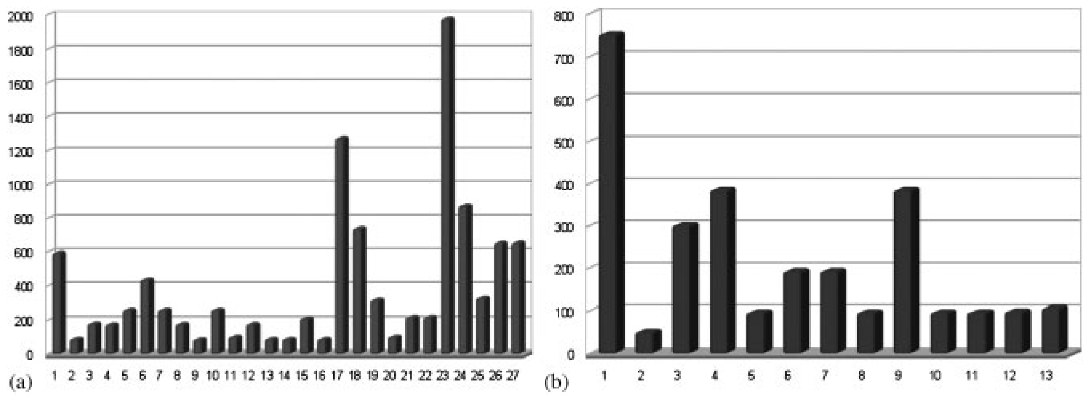

Figure 9. Basic Subversion operations during: (a) Phase 1 and (b) Phase 2.

Figure 9 shows the number of basic Subversion operations executed by the align procedure in SubCollaboration in the two phases of this case study.

The left side of the figure shows the basic operations to manage the 109 workspaces. These operations account for the basic Subversion commands ‘add’, ‘propset’, and ‘merge’. The total number of alignments done was 28 instead of 27 shown in the graph. The first alignment (in which only 74 of the 109 workspaces were created) executed a total of 6438 operations and was removed because the scale rendered the graph unreadable. All alignment operations required more than 80 basic Subversion operations, and the average number of basic operation per alignment is 608.82 distributed as 44.58% ‘add’ operations, 49.04% ‘propset’ operations, and 6.36% ‘merge’ operations. The large variation in these numbers is due to the magnitude of the changes in the reference workspace. Every change in a file requires a merge operation to propagate the change to the student workspaces. Some alignments included changes in files already present in the reference workspace plus the addition of new resources. Three of the alignments (excluding the initial one) required more than 800 basic operations. These reflect the addition of a large number of new resources to the repositories.

The right side of the figure shows the same data for the second phase in which 48 workspaces were created (students worked in groups of 4). A total of 13 alignment operations were executed. As in Phase 1, the most intensive alignment is the first one where the workspaces are created from scratch. Only one of the alignments required less than 80 basic operations. The number of operations in this phase is smaller than in phase 1 because the number of workspaces is also smaller, and the project required most of the documents to be contained in the workspace from the beginning. The distribution of the 2840 basic operations in this case is 32.99, 48.45, and 28.20% for the ‘add’, ‘propset’, and ‘merge’ operations, respectively.

Another measure of the complexity of the deployed repository is the size of the authorization file (see Section 3.3). The two phases required files with 3401 and 1060 lines, respectively, both sizes beyond the scope of a manual creation process. Merging conflicts only appeared in three of the alignment procedures and involved no more than three workspaces. The changes were manually merged with almost no effort.

With the previous numbers, it is clear that the number of operations performed using SubCollaboration is significantly less than the operations required to deploy such a scheme manually.

8. CONCLUSIONS AND FUTURE WORK

SubCollaboration, a platform to allow the use of the VCS Subversion over a large number of workspaces in collaborative environments has been described. The objective of this work was twofold. First, from the pedagogical point of view, students need to acquire the skills to manipulate tools that are regularly used in collaborative industrial scenarios. Second, provide a tool that supports the management of large numbers of team workspaces synchronized with a reference workspace designed by the teaching staff.

These objectives have been validated with two case studies producing the following results:

- Students require a small effort to learn to use VCSs.
- Version control is suitable for exchanging documents in a collaborative environment that includes the teaching staff
- The large number of administrative operations required to manage a large number of workspaces requires a significant level of automation.
- SubCollaboration supports this process hiding the low-level basic Subversion operations and providing a higher level view of the repositories to the instructors.

The use of version control to underpin collaborative work opens the door for several future avenues to explore. The regular exchange of information may be extended from regular files to activity traces to increase awareness among teammates. Also, team activity can be an excellent predictor of academic performance, thus anticipating problems and their solutions. In summary, from the results obtained so far, the potential of a closer monitoring of team activity can be beneficial for both the students and the teaching staff.

ACKNOWLEDGEMENTS

This work was partially funded by the Learn3 project, 'Plan Nacional de I+D+I TIN2008-05163/TSI', the Best Practice Network ICOPER (Grant No. ECP-2007-EDU-417007), the Flexo Project 'Plan Nacional de Investigación Científica, Desarrollo e Innovación Tecnológica' (Ref. TSI-020301-2008-19), and the 'Emadrid: Investigación y desarrollo de tecnologías para el e-learning en la Comunidad de Madrid' project (S2009/TIC-1650).

REFERENCES

1. Association for Computing Machinery. *Computer Science Curriculum 2008*: *An Interim Revision of CS 2001*. IEEE Computer Society: Silver Spring, MD, 2008.
2. Cabot J, Wilson G. Tools for Teams: A Survey of Web-based Software Project Portals. Dr. Dobb's, October 2009. Available at: http://www.ddj.com/development-tools/220301068 [last accessed December 2010].
3. Reid K, Wilson G. Learning by doing: Introducing version control as a way to manage student assignments. *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*. ACM: St. Louis, MO, U.S.A., 2005; 272–276.
4. Linder S, Abbott D, Fromberger M. An instructional scaffolding approach to teaching software design. *Journal of Computing Sciences in Colleges* 2006; **21**(6):250.
5. Collins-Sussman B, Fitzpatrick B, Pilato C. *Version Control with Subversion*. O'Reilly Media Inc.: Cambridge, MA, U.S.A., 2004.
6. Koschmann T. *Paradigm Shifts and Instructional Technology*: *An Introduction*, vol. 116, ch. 1. Lawrence Erlbaum: Mahwah, NJ, 1996; 1–23.
7. Stahl G, Koschmann T, Suthers D. *Computer-supported Collaborative Learning*: *An Historical Perspective*, vol. 2006, ch. 24. Cambridge University Press: Cambridge, 2006; 409–426.
8. Dillenbourg P. *What Do You Mean by 'Collaborative Learning'*?, vol. 1, ch. 1. Elsevier: Oxford, 1999; 1–19.

9. Lee B, Narayanan NH, Chang KH. An integrated approach to distributed version management and role-based access control in computer supported collaborative writing. *Journal of Systems and Software* 2001; **59**(2):119–134. DOI: 10.1016/S0164-1212(01)00013-9.
10. Spacco J, Hovemeyer D, Pugh W. An Eclipse-based course project snapshot and submission system. *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology EXchange*. ACM: New York, U.S.A., 2004; 56. DOI: 10.1145/1066129.1066140.
11. Hartness K. Eclipse and CVS for group projects. *Journal of Computing Sciences in Colleges* 2006; **21**(4):222.
12. Concurrent Version System. Available at: http://www.nongnu.org/cvs [last accessed December 2010].
13. Liu Y, Stroulia E, Wong K, German D. Using CVS historical information to understand how students develop software. *Proceedings of the International Workshop on Mining Software Repositories*, Edinburgh, Scotland, 2004; 32–36.
14. Mierle K, Laven K, Roweis S, Wilson G. Mining student CVS repositories forperformance indicators. *Proceedings of the 2005 International Workshop on Mining Software Repositories*. ACM: St. Louis, MO, 2005; 5.
15. Glassy L. Using version control to observe student software development processes. *Journal of Computing Sciences in Colleges* 2006; **21**(3):106.
16. Miura M, Kunifuji S. Development and practice of programming learning course management system with version control software. *International Conference on Knowledge*, *Information and Creativity Support Systems*, Japan, 2007; 152–159.
17. Lass RN, Cera CD, Bomberger NT, Char B, Popyack JL, Herrmann N, Zoski P. Tools and techniques for large scale grading using Web-based commercial off-the-shelf software. *ACM SIGCSE Bulletin* 2003; **35**(3):168. DOI: 10.1145/961290.961558.
18. Laadan O, Nieh J, Viennot N. Teaching operating systems using virtual appliances and distributed version control. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education—SIGCSE '10*, Milwaukee, WI, 2010; 480. DOI: 10.1145/1734263.1734427.
19. GIT. The fast version control system. Available at: http://git-scm.com [May 2010].
20. Helmick M. Integrated online courseware for computer science courses. *On Innovation and Technology in Computer Science* 2007; **39**(3):146. DOI: 10.1145/1269900.1268828.
21. Meneely A, Williams L. On preparing students for distributed software development with a synchronous, collaborative development platform. *ACM SIGCSE Bulletin* 2009; **41**(1):529–533. DOI: 10.1145/1539024.1509047.
22. National Survey of Student Engagement (NSSE). Engaged Learning: Fostering Success for All Students. Annual Report 2006, Available at: http://nsse.iub.edu/NSSE 2006 Annual Report/docs/NSSE 2006 Annual Report.pdf [last accessed December 2010].
23. Association For Computing Machinery. *Computer Engineering 2004. Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering*. IEEE Computer Society: Silver Spring, MD, 2004.