Coaching Asynchronous Teams for Undergraduate Programming Projects

Ken T. N. Hartness
hartness@shsu.edu
Computer Science Dept., Sam Houston State University
Huntsville, TX 77341, USA

Abstract

Team collaborations are used both in commercial, open-source, and academic software projects. More experience with team projects may be possible if intermediate students can receive guidance in using tools that support collaborations without requiring excessive class and instructor time to do so. This article makes a case for instructing students in the use of tools commonly used to support collaborations, including open-source projects with fewer opportunities for face-to-face meetings, and outlines a coach that aids the novice team member in using these tools. In this way, the student gains valuable skills for teamwork and in using tools commonly used to support teamwork.

Keywords: group projects, agents, coaching

1. INTRODUCTION

Many technical undergraduate degrees include at least one class that requires students to work together as a group. Given that many jobs related to their field of study require successful integration within a team collaborating towards a common goal, students can certainly benefit from greater exposure to collaborative team projects. Team collaboration may encourage social skills needed for individuals to work together, enhance the learning experience (Cronholm and Melin, 2006; Sloffer et al, 1999), make it possible for students to experience work on a more interesting project than they could tackle alone, and expose students to a problem that requires good design practices. However, team-based projects are difficult to coordinate and can lead to frustration from better students who worry about unrecognized efforts and ordinary students who feel inferior (Cronholm and Melin, 2006). These team-based projects are difficult to evaluate if the instructor wishes to assign grades based on individual effort, and some instructors are concerned about "deadbeat" students who learn nothing from the experience while relying on other team members to do the work.

Source code management systems are commonly used to support team software development. This alone is a good reason for students to be exposed to such systems. However, these tools are sometimes used in conjunction with communication support (e-mail being the simplest) to support development by people in geographically separated areas and, as such, may be of great help in allowing students with different schedules and living both on- and off-campus to collaborate more easily. In addition, these tools maintain logs of individual submissions and do not necessarily require that the instructor's first exposure to the students' work wait until final submission. On the other hand, many instructors feel that only advanced students can handle these professional tools, and, even then, they require considerable class or lab time to educate the students in their proper use.

The following sections describe the advantages and disadvantages of using source code management tools for team projects and outline an automated coach that seeks to address some of the disadvantages.

2. SOURCE CODE MANAGEMENT OF STUDENT PROJECTS

Gartner, Inc., a company specializing in research and advise related to information technology (Gartner, 2007) has suggested that successful IT programs must learn to support, among other things, tools for distributed, often asynchronous, collaboration, both in terms of tracking product as well as social interaction throughout the collaboration. They have suggested that over half of an individual's work will depend on input from other team members, perhaps dispersed geographically with team members in different countries and time zones (Grigg, 2001; Dwyer and Malani, 2006). Given that one of students' biggest complaints against team-based projects is the need to physically meet given disparate schedules and, sometimes, a lack of conveniently accessible meeting locations, training students in similar collaborative strategies makes sense.

Although an instructor could certainly argue that an academic program is not necessarily responsible for anticipating and providing all training required for a student to be successful in information systems careers, collaborative tools can provide other benefits to the learning process besides simply training a student in their use. Such tools can make it easier for students to work as a team without necessarily meeting physically, they can maintain a record of individual contributions, and an instructor can easily be included in the definition of a team and offer advice prior to the final submission date.

While a number of useful collaborative tools exist, Subversion (2006) and CVS (Harper, 1999) are open-source systems available for free download and are in common usage in open-source and even commercial software development. All further discussion of source code management will assume the CVS model. Files shared by the team are stored in a shared repository accessible over a network. Team members may check out copies of files, modify them, and submit the modified copy to the repository (Figure 1). Files are not locked by individual users as users routinely make changes to independent sections of files without conflict. If the modification conflicts with another team member's submission, the team member should determine how to handle this conflict and, then, resubmit the file. Tools are provided in

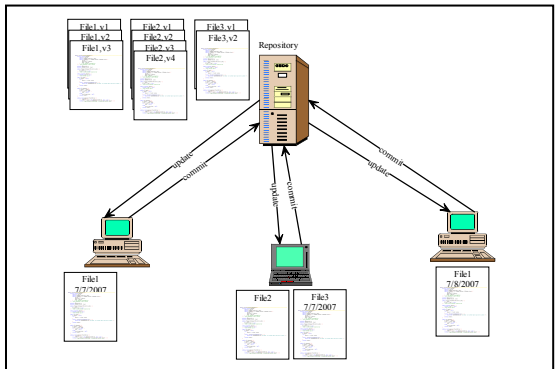

Figure 1 - Source Manager Server

the client software to recognize and locate these conflicts. A history of changes is maintained on each file, allowing earlier versions to be recovered. Any member can analyze this history to gain an understanding of individual participation in terms of number and size of contributions. CVS began as a collection of tools for updating local copies, committing changes from the local copy to the repository, comparing the contents of files, both local and in the repository, etc. Subversion is an attempt to address certain limitations of CVS.

Reid and Wilson (2005) used CVS as an assignment submission system. The shared repository makes it easy for students to move from one machine to another without physically carrying current copies of their work. Instructors and teaching assistants were able to assist students with problems while viewing a current copy of the student's work. Patterns of behavior, like waiting until the night before to begin an assignment, could be observed, and relevant suggestions could be made. In some cases, the history stored in a source code management system can be used as evidence in cases of suspected plagiarism.

Team projects are more easily supported. Students know where the most up-to-date versions of their teammates' work may be found. File comparison tools and comments logged during a commit of changed files allow students to discover what their teammates have changed. If an instructor or teaching assistant is included in each team, the team may even receive unsolicited advice as the work is being done, potentially resulting in a deeper level of instruction than is possible in an environment limited to lecture and evaluation of completed work

(Glassy, 2006). In any case, a deeper level of evaluation should be possible given that progress can be monitored in real-time or reconstructed from historical data.

Brereton et al (2000) were able to successfully manage student teams composed of students from different universities, and the indirect communication implied by interaction through the shared repository encouraged and guided explicit communication between the students. Their work indicates that a shared repository clearly reduces the significance of geographical collocation between team members. The same historical version information that could be used for evaluation of individual participation also helps team members remain aware of each others' activities; combined with e-mail or more sophisticated communication support, students are able to work together asynchronously with fewer face-to-face meetings. Superior support for asynchronous communication and tools that interpret CVS or Subversion logs in order to better visualize member activities will undoubtedly improve team members' ability to function asynchronously. Face-to-face status meetings, at least, become unnecessary if interested parties can simply view a summary of changes logged within the source code management system.

These advantages are easily eliminated by team members who fail to utilize the source code management system properly. If the repository is only used as a submission system, the history logs will be useless for determining individual participation. Instructors and teaching assistants can encourage use of the repository by refusing to assist students with work that is not accessible through the repository and, possibly, by evaluating progress points prior to the due date. A student who fails to update her work may prevent other team members from testing their own work in a timely manner; while this is nothing new for group projects, it is particularly frustrating in this case if a student is working diligently but forgetting to share her work with the others, even though it is easy to do so.

3. SOURCE CODE MANAGEMENT PROTOCOLS

Some instructors are naturally concerned about using valuable class time to teach students how to properly use a source code management system. Reid and Wilson (2005) claim to have spent one session on team work and version control and provided students with a tutorial on the details of using CVS (or another version control system used to manage source code). A single class may suffice to describe team programming, at least in a course where team programming is a means for gaining experience with other material and not an end in itself. The time spent training students to use a specific tool will vary depending, at least in part, on the availability of a good tutorial. By integrating a coach into the software development environment, I hope to include, among other things, the elements of a good tutorial in a tool that monitors day-to-day usage and provides context-sensitive help as a student needs it.

Students should be encouraged to follow a consistent protocol for incorporating the source code management system into their development process. Trytten (2005) ensured proper use of the system by automating it. Every file saved was committed to the repository. The following suggested protocols assume greater flexibility. Students should log their progress frequently until they are ready to test a module. This behavior helps eliminate any assumptions of inadequate participation on the part of other team members or an observing evaluator. However, once modules approach a point where they can be tested, several final modifications may be necessary before a collection of files are ready to be tested together, so students may prefer to wait to share all related files as one fully tested whole. Certainly, further modifications to files previously thought complete should probably be made locally until completed and tested so as to prevent temporarily replacing working code with incomplete changes that do not work. At this point, committed changes cease to be simply a log of gradual progress and become atomic steps forward from one (perhaps, partially) working solution to another (hopefully, superior) solution.

Begin team project work session:

1. Synchronize with shared repository.
   a. View files modified since last update from repository.
   b. Familiarize yourself with how these files have been changed. If the file is immediately important to your work, consider comparing it with your local copy for a detailed list of changes; otherwise, trust the comments logged by the user to summarize changes.
2. If the changes do not conflict with your own work, update the local copies; otherwise, communicate with other users, if necessary, and determine how to repair the conflict.
3. If you forgot to commit changes at the end of the last session, commit all changes ready to be shared with the team.

Create a new file:

1. If file needs to be shared with the team, check it into the repository.
2. If file contains a vital component that other members need, you may even wish to send them a message announcing its availability.

Save changes to a file:

- If file has never been tested and is not yet ready for testing, or another is responsible for testing it, commit the changes to the repository, immediately (otherwise, you may work until you are out of time and put off committing to the repository).
  a. Perform a file comparison between your local copy and the repository if you are not clear on all of the changes that you have made.
  b. Commit the file.
  c. Add a log comment that summarizes your changes so other users can quickly determine how the file has changed.
  d. If your changes conflict with changes committed by another member since your last update, communicate with other members, if necessary, and attempt to resolve the conflict (this may require waiting until later to complete the commit operation).

Complete successful testing of module(s) or end team project work session:

- If relevant files have not been committed, initiate commit operations for all files involved in the testing (follow same procedure for each file as above).

4. COACHING THE STUDENTS

4.1 Overview

The coach is an advising agent that monitors its owner's activities and shares this information with other coaches in a team. Whenever the coach determines that its owner needs some advise, it interrupts the owning user or alters one of the views visible to the user, depending on the nature of the advise. The coach combines the attributes of a version control assistant with those of a group awareness agent similar to Palantír (Sarma, 2003). Inexperienced users of CVS or Subversion will be assisted both in the proper use of the tool and in tracking their fellow group members' activities.

The coach attempts to model users in order to gauge the type and appropriateness of an intervention. Initially, the coach is quick to suggest that a student keep local copies of files synchronized with the shared repository files. It gradually learns to identify a work pattern, judge the experience level of the user, and use this information to adjust the likelihood and nature of an interruption.

The coach can encourage users to consider certain behaviors through a range of methods. These include marking a file that needs to be synchronized with the repository or displaying a warning in the status bar of a window. In important cases where this type of behavior is ignored, the coach can bring a warning box to the front or even steal the user's focus, truly interrupting their work in order to ensure that they see the advice. Naturally, this more extreme behavior is reserved for beginners who ignore other methods.

4.2 Implementation

Students access the shared repository via a modified shell or IDE (the author uses Eclipse [Eclipse.org, 2007] because of its extensibility). The subversion coach is initiated as a separate thread and tracks new resource creation, opening an existing re-

source, and modifying or saving changes. Common subversion commands are extended to keep the coach informed of check-outs from the repository as well as project synchronization commands to commit changed files to the repository and update local files from the repository.

File monitors track which files are open, how much they have been changed, and whether they have been committed back to the repository or otherwise synchronized with changes in the repository. Their findings are a major part of the data used by the coach to decide when to intervene. The data can also be communicated with other coaches so that a user's coach can anticipate conflicts and advise its user accordingly.

The coach uses a simple rule base to decide when to intervene with its user. The rules reflect the protocols described in the previous section. However, the rules are augmented to be triggered at different times for different users. Also, rules are included that reflect a philosophy of group awareness.

For example, another member of the group opens a file and makes changes without committing those changes to the repository. The user initiates a group session, starting the coach who recognizes that the user is a novice and suggests synchronizing with the repository and verifying any recent changes. The user then opens the same file being modified by the other member of the group. The icon for the file in the project window is modified to indicate it is being used by another user and a message appears in the status window suggesting that the user could communicate with the other member in order to avoid complicated merging of the two versions at a later time. The user ignores this and makes modifications to the file.

The coach informs the other member's coach, but the other member is experienced with no history of strong proprietary responses, so her coach chooses to simply augment the file icon in the project window to reflect that someone is working on the same file.

A third member, however, has already committed extensive changes to the file and has a history of proprietary behavior such as quickly viewing a modified file of interest and subsequently making further modifications to the file. At present, the coach would assist this user's desire to keep informed of changes to this file and warn its user. Later versions might explore methods that may assist proprietary users to view the work as a team effort.

4.2.1 User model. A user model records the user's experience level, willingness to be interrupted, current working pattern, degree of involvement with the different files belonging to the group, tendency to respond to file modifications by others, and a history of the last time the coach advised the user on a particular topic.

Experience levels reflect novice, intermediate, and expert experience with the coached group environment.

Current working patterns include incremental and version-oriented. The current working pattern is primarily used to augment responses for more experienced users, suggesting expectations for user behavior. Incremental working patterns reflect a tendency to associate the commit action with saving a file or the end of a work session; every modification is committed to the shared repository soon after it is made, creating an up-to-date public record of the user's activity as well as providing a backup of current work. This behavior is useful at the beginning of a project so that students have some awareness of the progress of the group as a whole. Once parts of a module are ready for testing, a user may choose a version-oriented working pattern in which modifications are only shared when they have passed preliminary tests (or, at least, are believed capable of doing so). The coach does not pester users to commit all changes if it believes they are following the version-oriented pattern.

The measure of user involvement with a file is determined by the proportional amount of changes made by the user to the file and the intensity of their response to changes made by others (latency between modification and the user viewing the change, and whether the user immediately made changes as well). In addition, a record is made of the type of response made by the user when a file of interest to the user is changed by another.

The history of past advice is used to prevent the coach from pestering users, especially

the more experienced ones, by repeatedly offering the same advice. Delays between one offer of advice and the next are related to the experience level of the user.

The coach maintains a persistent model of its user utilizing the same repository as the group project. A coach is created in connection with a particular repository, group name, and user name. A folder is saved in the repository with the same name as the group and the text ".grp" appended to the end. The coach updates its local copy of this folder and modifies it to reflect its owner's current IP address for peer-to-peer communication between active coaches. A sample folder is shown in figure 2. Each file in the folder contains the user's long name, current IP address (or zero), a list of modified files since last commit, a list of committed files that haven't been locally updated, and the serialized data of the user model. Since each coach maintains its own models of team members using identical algorithms, it is sufficient for a user to save his or her own model for all to share. In a system of heterogeneous coaches, a different method would be desirable.

4.2.2 Rule-based system: The rule base is consulted whenever any repository synchronization is attempted, after a changed resource is saved or committed, and after receiving a change notification from one of the file monitors.

Rules indicate behaviors such as the following:

> If commit of resource by another and resource is important to owner then encourage synchronization with the repository based on importance.
>
> If owner is requesting a commit of a resource that has been committed by another, then encourage synchronization with the repository (top priority) and describe merging.
>
> If resource modified by owner that is very important to someone else, encourage a commit immediately (medium priority).

Each rule is primarily triggered by an event. Recognized events include

- ACTIVE_USER
- OPEN_RESOURCE
- LOCAL_UPDATE,
- CHANGE_NOTIFICATION
- GLOBAL_UPDATE
- INACTIVE_USER

ACTIVE_USER (remote message) includes a user's name and IP address. It simply adds a user's IP address to the list of active group members. The coach also sends the contacting coach a list of open files since it may not have been active when OPEN_RESOURCE events were sent to the group, initially.

OPEN_RESOURCE includes the name of the user opening the resource and the name of the resource. Local events are shared with other active coaches. A resource monitor is established to track the degree to which a resource is modified. If the event is generated remotely, the resource's icon is updated in the explorer view. If the user's model indicates that the resource is important to the user and the user would want to be informed of changes, advise the user that the resource is being modified, remotely.

CHANGE_NOTIFICATION includes a user's name, the resource's name, and a measure of how much the resource has been modified. If generated locally, beginners are encouraged to commit changes if the degree of change is sufficiently large. Other levels are also encouraged to commit changes if the work pattern is incremental, although the degree of change must be larger before the coach gets involved. A remote version of this message is combined with the user's interest in this resource, whether the user is currently modifying the resource, as well, and the user's tendency to respond to another user's modification of an interesting resource in order to decide whether to inform the user and, possibly, suggest that the user contact the other user in order to coordinate changes to the resource.

LOCAL_UPDATE indicates that a changed resource has been saved but not yet committed to the repository. Beginners and users with incremental work patterns may be encouraged to commit changes, depending on level of user and time since coach last advised user in this way. A remote message of this form is handled in much the same way as a remote CHANGE_NOTIFICATION.

GLOBAL_UPDATE indicates that a changed resource has been committed to the shared repository. If local, the message is shared

with other active coaches. If remotely generated, then factors similar to those mentioned in the previous paragraph are used to decide if the user needs to be notified and possibly encouraged to synchronize in order to get the most up-to-date version of the resource.

Factors such as the user's willingness to be interrupted, the user's skill level, and the time since a similar piece of advice was issued are considered before a user is interrupted or distracted by advice.

5.CONCLUSIONS

Students need experience with teamwork, especially in business and technical fields. Group software projects are often difficult to coordinate and evaluate, but CVS, Subversion, and similar source code management systems may offer ways to manage student projects while teaching students valuable skills. I have introduced the concept of a coach, by itself not a new idea, for guiding novice users of source code management systems in their proper usage. The coach should also include conflict awareness and provide tools for helping team members remain aware of one another and encouraging them to avoid or resolve conflicts.

In the future, I hope to add a chat tool that links with a source code development system and doubles as a log of conversations as well as a messaging system for asynchronous communication, radar views that summarize student activities and the degree to which files have been changed, and changes to the coach so that it knows when it is appropriate to bring up these tools or encourage its user to do so.

6. REFERENCES

Brereton, O., S. Lees, R. Bedson, C. Boldyreff, S. Drummon, P. Layzell, L. Macaulay, and R. Young (2000). Student collaboration across universities: A case study in software engineering. *Thirteenth Conference on Software Engineering Education & Training*.

Cronholm, S., and U. Melin (2006). Project oriented student work: Group formation and learning. *Proceedings of the Information Systems Education Conference (ISECON 2006),* Dallas, TX, USA.

Dwyer, C., and P. M. Malani (2006). Low-cost collaborative tools for virtual communication. *Information Systems Education Journal* 4 (78).

Eclipse.org (2007). Eclipse.org home. URL: http://www.eclipse.org. Copyright 2007 by The Eclipse Foundation.

Gartner, Inc. (2007). Gartner delivers the technology-related insight necessary for our clients to make the right decisions, every day. http://www.gartner.com.

Glassy, L. (2006). Using version control to observe student software development processes. *Journal of Computing Sciences in Colleges* 21 (3).

Grigg, J. (2001). *The Uncertain Future: Technology and Business Challenges for IT.* The Gartner Group.

Harper, D (1999). cvsnt - Concurrent Versions System 2.0.51d, WinCvs 2.0.2.4, open source software published at www.wincvs.
org (downloaded 9/1/2005).

Reid, K., and G. Wilson (2005). Learning by doing: Introducing version control as a way to manage student assignments. *ACM SIGCSE Bulletin* 37 (1).

Sarma, A. (2003). Configuration management workspace awareness for distributed software development. *European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE) Doctoral Symposium,* Helsinki, Finland, September 2003.

Sarma, A., A. van der Hoek, and D. Redmiles (2007). A comprehensive evaluation of workspace awareness in software configuration management systems. *IEEE Symposium on Visual Languages and Human-Centric Computing,* Coeur d'Aléne, Idaho, September 2007.

Sloffer, S. J., B. Dueber, and T. M. Duffy (1999). Using asynchronous conferencing to promote critical thinking: Two implementations in higher education. *Proceedings of the 32nd Annual Hawaii International Conference on System Sciences,* Wailea Maui, Hawaii, USA, 5-8 January 1999.

Subversion Home (2006). http://subversion.tigris.org, Collab-Net.

Trytten, D. (2005). A design for team peer code review. *ACM SIGCSE Bulletin* 37 (1).