AC 2008-2218: OPEN SOURCE SOFTWARE TO SUPPORT STUDENT TEAMS: CHALLENGES, LESSONS, AND OPPORTUNITIES

Clifton Kussmaul, Muhlenberg College

Clifton Kussmaul is Assistant Professor of Computer Science at Muhlenberg College, and Chief Technology Officer for Elegance Technologies, Inc. He has a PhD from the University of California, Davis, an MS and MA from Dartmouth College, and a BS and BA from Swarthmore College. His interests include agile development, virtual teams, entrepreneurship education, and cognitive neuroscience, particularly auditory processing.

Open Source Software to Support Student Teams: Challenges, Lessons, and Opportunities

Abstract

Team projects have a long history in education, with an extensive literature. Appropriate tools and procedures can support team projects, and open source software tools present specific opportunities and challenges. Open source software (OSS) generally refers to software that is distributed without charge and with the original source code, so that anyone can fix defects, add enhancements, or otherwise modify the software and share their changes with others. Thus, OSS can be freely installed on any number of computers, and modified by faculty and students with appropriate knowledge, but it may include less documentation, require more expertise to install and maintain, and be more difficult to evaluate than commercial alternatives. Since team projects usually involve multiple activities, teams can use multiple tools and try to make them work well together, or teams can use tools that support multiple functions. We focus on tools that can support collaboration (sharing documents and information) and coordination (keeping track of which team members are doing what tasks). In particular, we review previous work and describe recent experiences using wikis, version control systems, task tracking systems, and combinations of these tools. We describe key features, effective practices, supporting activities and assignments, and student outcomes. We also summarize best practices, lessons learned, and directions for future experimentation and development. Using OSS tools helps students learn to use new tools, exposes them to tools or types of tools they are likely to encounter in the future, and enables them to attempt and complete more ambitious projects under more realistic conditions. Like any tools, OSS requires an ongoing time investment by faculty, but helps them to diagnose and correct problems, assess student performance, and help the projects and teams adapt to other factors.

1. Introduction

Open source software (OSS) is distributed without charge and with the underlying source code, so that other software developers can fix defects, update documentation, add enhancements, or otherwise modify the software and share the changes with others. Most OSS projects consist of a small core team of developers, and a broader community of people who use the software, report defects, and support the project in other ways. As of October 2007, SourceForge.net hosted over 150,000 OSS projects, although many of them are small and have little activity.

For student team projects, OSS offers both benefits and risks. It can be freely installed on any number of computers, and it can be modified by faculty, staff, and students with appropriate knowledge. (This can even serve as the focus for software engineering courses[1].) OSS can be more difficult to evaluate than commercial alternatives, which place more emphasis on sales and marketing, although the more popular projects are often reviewed online, in trade publications, or academic venues. OSS may include less documentation and require more expertise to install and maintain, although again the larger, more popular projects recognize these challenges and work to address them.

Team projects have a long history in education, with an extensive literature[2,3,4] which this paper does not attempt to summarize. However, team activities (and supporting tools) can be classified according to their broad objective(s):

1. **Cohesion**. Teams need to develop and maintain cohesion - a sense of community, trust, and shared purpose. Tools can only indirectly support cohesion.
2. **Communication**. Teams need to communication information. Relevant tools (e.g. email, instant messaging, and text messaging) are commonplace, and free or inexpensive.
3. **Collaboration**. Teams need to collaborate on a variety of documents. Several types of tools can be very helpful.
4. **Coordination**. Team members need to know what tasks need to be performed, how important they are, who is working on which tasks, what resources (e.g. people, time, money) each task will take, and when each task must be completed. Again, appropriate tools can facilitate such coordination.

The rest of this paper focuses on how OSS tools can be used to support team projects, including:

- Ways in which tools can support team projects.
- Selecting appropriate tools.
- Learning to use tools effectively.
- Assessment and reflection.

2. OSS Tools

OSS tools can support specific team activities. However, teams usually perform a variety of activities, which could benefit from different tools. Teams can approach this situation in two ways. First, they can use a tool that provides multiple functions, and try to adapt it to multiple activities. Second, they can use multiple tools, and try to make the tools work well together. The following sections describe some specific functions and tools, and then ways of combining them.

2.1. Version Control

Team members need reliable access to the most current version of a document, and they may also need access to earlier versions of the same document. When several people are working on the same document, they need to prevent or resolve conflicts, and be able to identify who made which changes. Informal approaches may be sufficient for small teams, but not for large teams. Software developers have addressed such challenges for decades with *version control systems (VCS)*. A VCS typically saves all versions of every document, and logs the date, author, and a description of each new version. This makes it possible to retrieve previous versions and to review the history and evolution of a document. A VCS may also provide ways to lock documents so that other people can't edit them, as well as ways to merge different versions. There are a variety of VCS's, but the most common OSS systems are Concurrent Versions System (CVS) and SubVersion (SVN). VCS's are usually designed to work with plain text files, but they can also handle other file types. For example, we use SVN repositories for software source code, web sites, academic papers, teaching notes, examples, and handouts. A number of authors discuss ways to incorporate VCS's in courses and extract useful data from them[4-9].

2.2. Wikis

A *wiki* provides many of the benefits of a VCS in a form that is easier for non-programmers to use. A wiki is a special web site that allows any user to create and edit pages, and link pages together, without specialized tools or knowledge[10, 11]. There are many different wiki systems, each with strengths and weaknesses[12-14]. Typically, pages are edited as plain text with simple formatting conventions, although some wikis include graphical text editors. This makes it easy to highlight key ideas or comment on other people's work. Wikis also allow users to see a page's history, including all previous versions, and which changes were made by whom. The history also allows students and faculty to review and reflect on how documents evolve over time. Several authors review ways to use wikis for team projects and to analyze outcomes[15-18].

The best known wiki is probably Wikipedia, a free-content encyclopedia with over five million articles in over 250 languages. However, wikis can also be used in other ways for a variety of creative and educational purposes[19-25]. We have worked with several wiki systems. MoinMoin and PmWiki are easy to install and configure, and are flexible enough for a variety of smaller projects. MediaWiki is used for Wikipedia, but we have found it more difficult to adapt to other purposes. TWiki is widely used in corporate intranets, and has a wide variety of plugins to add specialized functionality, but may be too complex for simple projects. The Moodle course management system and the Trac project management system both contain wikis which lack some common capabilities, but contain some special features; for example, the Moodle wiki can easily be configured to support individual, group, or whole class assignments, while the Trac wiki can automatically link to the VCS and the task tracking system (see below).

2.3. Document Management

D*ocument management systems* (DMS) are also designed to help teams manage documents. A DMS is particularly useful for workflow, where documents go through specific revision and approval stages. There are numerous proprietary and OSS DMS's, but we have not used them with student teams.

2.4. Task Tracking

For small, co-located teams with relatively simple projects, the team members may be able to coordinate their activities informally. A better approach is to use a VCS, wiki, or DMS (see above) with a shared document to track every task and its current status. For example, we often use a spreadsheet with a row for each task, and a column for each piece of associated information, such as priority, difficulty, current owner, start date, completion date, etc. This makes it easy to sort or filter tasks, and to do simple reporting. For larger, more complex projects, *task tracking* tools (both proprietary and OSS) are highly recommended. Bugzilla is a well known OSS task tracking system, intended specifically for defect tracking. The Trac project management system includes a task tracking system that works well for both tasks and defect reports. Task trackers can also provide useful data about team behavior[26].

2.5. Multifunction Tools

Tools that provide multiple functions are often easy to use, since the functions were designed to work together. For example, a course management system (CMS) such as the Moodle (OSS) provides a variety of functions designed to work smoothly together, and includes functions that can support team projects; this can be a good first approach for institutions that already have a CMS. Similarly, the Trac project management system combines SVN, a wiki, and task tracking; we use it for industry projects and student team projects. There are a variety of other *groupware* systems designed to support communication, collaboration, and coordination.

Although some multifunction tools are more difficult to install and maintain than individual tools, the total effort is usually less than the effort required to install, integrate, and maintain multiple tools. Another advantage is that there may be commercial hosting providers (this is true for both Moodle and Trac). A disadvantage of such tools is that their individual functions may be less sophisticated than in more narrowly specialized tools. For example, the wiki contained in the Trac project management system lacks features common to most other wikis, such as the ability to rename pages.

2.6. Multiple Tools

Tools designed for a specific function are often simpler and more effective to use, and are usually easier to install, maintain, and modify. Thus, faculty or students can choose the set of "best of breed" tools that best fit the current requirements and constraints.

However, using and integrating multiple tools presents a number of challenges, particular when three or more tools are involved. The team must learn to use each tool, and agree on which tool to use for which activities, particularly when the tools have overlapping capabilities. For example, are meetings scheduled via email, tickets, or a wiki page? What about customer requests? In 2006 we used the discussion forums and grading functions in Moodle, and the VCS, wiki, and task tracking in Trac. Supporting materials ended up scattered across both systems, and no one was sure where to find them. In 2007 we used Trac for everything; it doesn't do everything perfectly, but there was less confusion. Installing and maintaining multiple tools takes more effort, particularly if the tools need to share information (such as user names and passwords). It also requires more experience to evaluate, select, and integrate combinations of tools.

Sometimes, however, multiple tools can work well together. For example, Bugzilla and MediaWiki are more powerful and flexible than the corresponding functions within Trac, and there are detailed instructions for configuring Bugzilla, SubVersion, and MediaWiki to create a more complete environment[27]. These types of integrated systems will become more common, as OSS projects compete to provide more functionality with less effort.

3. Selecting OSS Tools

The general process for evaluating OSS tools is similar to the process for commercial tools, although some of the details are different. Note that this evaluation process can be a good experience for students or student teams. We recommend the following process:

1. **Identify your requirements and preferences**, and prioritize them as essential, desirable, or potential useful in the future. include factors such as:
    a. Language support, particularly if English is not the preferred language.
    b. Licensing options, particularly if the tools may be used for commercial activity.
    c. Technology: hardware, operating system, programming language, database.
2. **Identify and quickly assess candidate tools**. Start with general descriptions, including the tool's homepage, reviews in magazines, blogs, or other web sites, and sites specifically devoted to comparing tools in a category (e.g. WikiMatrix[13] or WikiEngineComparison[15]). In addition to the requirements identified in step 1, consider:
    a. Maturity: Is the tool ready for production use, or still a work in progress
    b. Frequency of new releases: look for tools with new releases every 6-12 months; very short cycles can be hard to keep up with, and very long cycles may be a sign that the tool is not actively being developed. Newer projects often release more frequently than more mature projects.
    c. Number of core developers and active users: while most OSS projects have a small team of core developers, a project with just one or two key developers could stall if they stop work. Healthy projects usually have a larger community of active users who can answer questions and help solve problems.
    d. Support options: Some larger OSS projects offer commercial hosting, support, or consulting assistance.
3. **Evaluate the best candidates in more detail**.
    a. Study documentation (if any) for users, administrators, and developers.
    b. Browse relevant discussion forums. Are there active contributors? Are questions answered promptly? How do disagreements get resolved?
    c. Try to download, install, and configure each tool that still seems feasible.
    d. Test your most important requirements, and a selection of your less important requirements. How hard is it to modify the tool, or deal with unexpected needs?

This process can help you to identify the most promising candidates, but it may take much longer to truly understand and appreciate the strengths and weaknesses of different options. As a result of the process outline above and experience using these tools with teams in academia and/or industry, we recommend the following OSS tools:

Type	Name	URL	Comments
VCS	Concurrent Versions System (CVS)	http://www.nongnu.org/cvs	widely used, good integration with other tools
VCS	SubVersion (SVN)	http://subversion.tigris.org	newer than CVS, good integration with other tools
wiki	PmWiki	http://pmwiki.org	easy to install & configure
wiki	MoinMoin	http://moinmo.in	easy to install & configure
wiki	MediaWiki	http://www.mediawiki.org	used for Wikipedia, but more difficult to adapt for other purposes
wiki	TWiki	http://twiki.org	powerful & flexible, but may be too complex for simple projects
DMS	KnowledgeTree	http://www.knowledgetree.com	haven't used with student teams; works well for industry teams
tasks	Bugzilla	http://www.bugzilla.org	
CMS	Moodle	http://www.moodle.org	course management system; good for small teams & projects
multi	Trac	http://trac.edgewall.org	combines task tracking, wiki, and SubVersion – works well for student & industry teams

Table 1: Recommended open source software projects

4. Learning to Use Tools

We recommend the following guidelines to help students learn to use new tools or functions.

1. Use a small set of tools, but as widely as possible. For example, although we use a course management system for all of our other courses, in software engineering we use the Trac project management system for posting handouts, making assignments, and other course management tasks, so that students don't have to switch between different tools.
2. Introduce them gradually, so that students (and teachers) aren't overwhelmed.
3. Introduce them "just in time", before students are about to use them in the course.
4. For more difficult tasks, consider a preliminary activity or assignment to give students a short, successful experience, before a larger, more demanding assignment.

For example, activity 1 (below) introduces students to Trac. We usually preview the activity on a projection screen and then give students a fairly terse set of instructions; otherwise, we might provide more detailed instructions, including screenshots.

Activity 1: Introduction to Trac

1. Go to the main course page in Trac [URL provided].
2. Login (using your usual username and password).
3. Explore a little bit - try each top level menu item to see what happens.
4. View the list of tickets.

5. Accept the ticket assigned to you.
6. Create a wiki page for yourself, using the link on the main course page.
   (WikiFormatting is a good reference.)
7. Update the ticket.
   1. Ask someone to review your wiki page.
   2. Explain anything in particular they should look for.
   3. Assign the ticket to the next person in the list of names.
8. View the list of tickets.
9. Accept the updated ticket assigned to you.
10. Review the wiki page.
    1. If you find problems, update the ticket and assign it back to the author.
    2. If you don't find problems, close the ticket.

Similarly, activity 2 (below) introduces students to peer review with a wiki. In some cases, it is preceded by an activity focused on basic wiki editing, depending on the students and on the wiki platform (some wikis have graphical editors similar to word processors, while other wikis use simple but unfamiliar markup languages).

Activity 2: Wiki Peer Review

1. Carefully read through the entire document at least once.
2. Mark what you understand as the most important ideas.
   (First reviewer use bold, second reviewer underline.)
3. Use colored text to indicate your inline comments and additions.
   (First reviewer use green, second reviewer use red.)
4. Use a strikethrough to indicate text you feel should be deleted.
   Add a comment to indicate why you are suggesting deletion.
5. Add links for lengthy comments.
6. At the end of the document, write a summary paragraph presenting an overarching and constructive critique.

5. Assessment

During Fall 2006 and 2007, the students in a software engineering project course were surveyed (at the end of the course) regarding the usefulness of the Trac project management system. Their responses are summarized in Table 2 below.

How would you describe your experience using Trac (SVN, tickets, & wiki) … 1=very negative, 5 = very positive	Fall 2006 (n=7)	Fall 2007 (n=7)
… overall	4.1	3.8
… to store & share information	4.1	4.1
… to communicate with other students	4.0	3.3

Table 2: End of course survey results

It is difficult to directly compare results from the two years. In small, project-based courses, the motivation and aptitude of a few students can have a big impact on the course. Also, the course organization changed significantly in the second year; for example, in 2007 the students worked on a series of projects, none of which were as large or complex as the projects in 2006.

Students also commented on the value of using Trac:

- Fall 2006
  - "I like being able to store info online."
  - "Easy to change things/update/store/share."
  - "It's easy for people on a team to communicate."
  - "It kept me on schedule of what to do."
- Fall 2007
  - "Easy to communicate with all group members and teacher with one post. Difficult to use at first but smooth after."
  - "Trac allowed for smoother collaboration. I will probably use Trac if I work in teams with software."
  - "Organizing and planning for projects was easier with tickets."
  - "[Trac] works well to organize large projects but for smaller ones, it just makes more busy work and doesn't help."

In some cases, however, students may not recognize the value of such tools and processes until several years later, when they encounter larger projects in school or in the workforce. A former student, working in industry, emailed to describe the effects of processes and tools on his career:

> "Even though I (we) complained during our software engineering class about the tedious paper work that was required for our projects, it was by far one of the most accurate representations of a 'real world' work environment. The processes that we underwent in your class to develop a project (especially scoping it out) have given me a leg up on my peers. I have come to understand that the (good) developers spend roughly 65% of their time planning, 20% testing and 15% coding. So thank you."

6. Lessons Learned & Recommendations

First and most importantly, remember that "the major problems of our work are not so much *technological* as *sociological* in nature"[28]. In other words, the biggest challenges tend to involve personalities and relationships between people. Tools can help a good team to function more effectively, but can't prevent a poor team from having problems, although in some cases tools can help diagnose problems.

Keep things simple, at least initially. Start with one or two simple tools that will make the biggest difference in the project or course. Develop experience and confidence, and then decide whether to add more tools, or switch to an integrated tool. Use integrated tools, or preconfigured "best of breed" sets, before trying to configure multiple tools to work smoothly together. Look for OSS projects that have a history of releasing new versions at least once a year, and have multiple developers and healthy user communities.

Explore tools and techniques gradually. We usually start with communication tools, which are the most familiar to students, and then introduce collaboration and coordination tools "just in time", for the course and project. This can be a challenge; students often recognize when a tool is unnecessary, but may have trouble learning a necessary tool in time to address a new challenge.

Finally, don't give up too quickly, and be ready for occasional failures. In some cases, failure can provide an opportunity for classroom discussion about the role of tools, and the importance of matching tools to specific situations. If every experiment is a success, then we may not be experimenting enough.

7. Conclusions

This paper has examined how OSS tools can be used to support team projects. It described several tool categories, along with a process for selecting and evaluating candidate tools. It listed a variety of tools that have proved useful in academic and industry settings, and described some activities used to introduce students to such tools. The paper reports results of student surveys, which are generally positive, although the small class size and differences between semesters limit the reliability and validity of the data.

Using OSS tools benefits students in several ways. First, they will be better able to attempt and complete ambitious projects. Second, they will be exposed to tools that they are likely to encounter in the future. Learning new tools requires time and effort that could be spent on other activities, but students also benefit by improving their ability to learn new tools. In some cases, however, students may not immediately see the value of tools.

Using OSS tools also benefits faculty. They make it easier to manage student teams, and adapt the projects and teams in response to other factors. The tools can also provide visibility into project and team dynamics, which can help faculty diagnose and correct problems, and assess student performance more accurately.

References

1. Petrenko, Maksym, Denys Poshyvanyk, Vaclav Rajlich, and Joseph Buchta. 2007. Teaching software evolution in open source. *IEEE Computer* 40, no 11: 25-31.
2. Tomayko, James E. 1987. *Teaching a Project-Intensive Introduction to Software Engineering*. Pittsburgh, PA: Carnegie Mellon University, Software Engineering Institute.
3. Tomayko, James E. 1998. Forging a discipline: An outline history of software engineering education. *Annals of Software Engineering* 6, no. 1 (March 1): 3-18.
4. Fincher, Sally, Marian Petre, and Martyn Clark. 2001. *Computer Science Project Work: Principles and Pragmatics*. Springer.
5. Clifton, Curtis, Lisa C. Kaczmarczyk, and Michael Mrozek. 2007. Subverting the fundamentals sequence: Using version control to enhance course management . *Proceedings of the 38th SIGCSE Technical Symposium on Computer Science Education*.
6. Glassy, Louis. 2006. Using version control to observe student software development processes. *Journal of Computing Sciences in Colleges* 21, no. 3: 99-106.

7. Hartness, Ken T. N. 2006. Eclipse and CVS for group projects. *Journal of Computing Sciences in Colleges* 21, no. 4 (April): 217-222.
8. Liu, Y, E Stroulia, K Wong, and D German. 2004. Using CVS historical information to understand how students develop software. *Proceedings of the 2004 International Workshop on Mining Software Repositories*.
9. Mierle, Keir, Kevin Laven, Sam Roweis, and Greg Wilson. 2005. Mining student CVS repositories for performance indicators. *Proceedings of the 2005 International Workshop on Mining Software Repositories*.
10. Reid, Karen L., and Gregory V. Wilson. 2005. Learning by doing: Introducing version control as a way to manage student assignments . *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*.
11. Lamb, Brian. 2004. Wide open spaces: Wikis, ready or not. *EDUCAUSE Review* 39, no. 5 (October): 36-48.
12. Leuf, Bo, and Ward Cunningham. 2001. *The Wiki Way: Quick Collaboration on the Web*. Addison-Wesley Professional.
13. CosmoCode. WikiMatrix - Compare them all. http://www.wikimatrix.org (accessed Oct 5, 2007).
14. Schwartz, Linda, Sharon Clark, Mary Cossarin, and Jim Rudolph. 2004. Educational wikis: Features and selection criteria. *The International Review of Research in Open and Distance Learning* 5, no. 1.
15. WikiEngineComparison - MoinMoin. http://moinmoin.wikiwikiweb.de/WikiEngineComparison (accessed Oct 5, 2007).
16. Kay, Judy, Nicholas Maisonneuve, Kalina Yacef, and Peter Reimann. 2006. The big five and visualizations of team work activity. *Proceedings of the 8th International Conference on Intelligent Tutoring Systems*.
17. Kay, Judy, Nicholas Maisonneuve, Kalina Yacef, and Osmar Zaiane. 2006. Mining patterns of events in students' teamwork data. *Proceedings of the Workshop on Educational Data Mining at the 8th International Conference on Intelligent Tutoring Systems (ITS 2006)*.
18. Korfiatis, Nikolaos, and Ambjörn Naeve. 2005. Evaluating wiki contributions using social networks: A case study on wikipedia. *Proceedings of the First On-Line Conference on Metadata and Semantics Research (MTSR'05)*.
19. Viégas, Fernanda B., Martin Wattenberg, and Matthew M. McKeon. 2007. The hidden order of Wikipedia. *Proceedings of the 12th International Conference on Human-Computer Interaction*.
20. Angeles, Michael. 2004. Using a wiki for documentation and collaborative authoring. *LLRX.com*, November 28. http://www.llrx.com/features/librarywikis.htm (accessed June 27, 2007)
21. Fountain, Renée. 2007. Wiki Pedagogy. http://www.profetic.org/dossiers/dossier_imprimer.php3?id_rubrique=110 (accessed June 27, 2007).
22. Kussmaul, Clif, Susannah Howe, and Simon Priest. 2006. Using wikis to foster team communication, cohesion, & collaboration . *Proceedings of the 2006 American Society for Engineering Education (ASEE) Annual Conference & Exposition*.
23. Kussmaul, Clifton, and Sharon Albert. 2007. Reading, writing, and revising with wiki technology: Tutorial presentation. *Journal of Computing Sciences in Colleges* 22, no. 6: 138-139.
24. Notari, Michele. 2006. How to use a wiki in education: 'Wiki based effective constructive learning' . *Proceedings of the 2006 International Symposium on Wikis*.
25. Read, Brock. 2005. Romantic poetry meets 21st-century technology. *The Chronicle of Higher Education*.
26. Wei, Carolyn, Brandon Maust, Jennifer Barrick, Elisabeth Cuddihy, and Jan H. Spyridakis. 2005. Wikis for supporting distributed collaborative writing. *Proceedings of the Society for Technical Communication 52nd Annual Conference*, May 8.
27. Liu, Chang. 2005. Using issue tracking tools to facilitate student learning of communication skills in software engineering courses. *Proceedings of the 18th Conference on Software Engineering Education & Training (CSEET '05)*
28. Segetech Ltd. 2007. Bugzilla/SVN/Wiki Integration. http://oss.segetech.com/bugzilla-svn-wiki.html (accessed Oct 5, 2007)
29. DeMarco, T. and T. Lister. 1999. *Peopleware: Productive Projects & Teams*. Dorset House.