Group Work Support for the BlueJ IDE

Kasper Fisker
fisker@fiskers.org

Davin McCall
School of Engineering and IT,
Deakin University
davmac@deakin.edu.au

Michael Kölling
Computing Laboratory,
University of Kent
mik@kent.ac.uk

Bruce Quig
School of Engineering and IT,
Deakin University
bquig@deakin.edu.au

ABSTRACT
Learning to work in teams is essential for every software professional. Developing software as a team project is the standard practice in industry, and should be practiced in university courses. Starting effective group work practices early can lead to better acceptance of group work as a standard development mode.

Nonetheless, group work is often not included in introductory programming courses. The reason is often the necessary overhead associated with developing software in groups. We present a design and implementation of group work support tools integrated into the educational BlueJ IDE, which remove much of the tool overhead and make it easier to include group work in introductory courses.

Categories and Subject Descriptors
D.2.9 [**Software Engineering**]: Management - Programming teams.
K.3.2 [**Computers & Education**]: Computer & Information Science Education - *Computer Science Education*

General Terms
Design, Human factors.

Keywords
Computing education, group work, support tools, BlueJ.

1. INTRODUCTION
For a well-educated computer science graduate, the ability to work in groups is an essential skill. Most educators and professionals agree that group work is a topic that should be explicitly taught and practiced in any computer science or software engineering undergraduate degree [7, 8, 9]. The benefits of getting students to work in groups early in the course – preferably in their first year – have been pointed out repeatedly in the literature [9, 3].

One of the most important aspects of starting group work early is to get students to embrace this way of working as the standard case and give them a chance to practice it repeatedly. Adding group work late is likely to be met with resentment in some students.

Despite this general agreement, group work is very commonly not handled well in contemporary introductory courses. The actual situations in classrooms show that many courses do not include any group work, and that a considerable number of those that do offer little organised support for this activity.

We speculate that this discrepancy – widespread agreement of the value of including group work contrasted with relatively limited inclusion of organised group work in first year courses – is a result of the considerable overhead that more explicit teaching of group work requires.

The problems that lead to this overhead are varied. Firstly, working in groups is a complex task that takes time to master. This is a characteristic that we cannot remove completely – some of the complexity is intrinsic to the problem. There are, however, a number of complexities stemming from secondary factors that we may well be able to reduce.

One of the most prominent problems is the lack of dedicated tool support for group work of beginning student programming teams. Students are often reduced to using email and chat sessions as their main support tools – technologies which do not offer adequate support for many of the problems at hand.

In the computer science literature, few discussions of group work support tools are found, and those that are available typically concentrate on code sharing tools for professionals as opposed to beginning programmers. One exception is the work of Čubranić and Storey [4] who discuss group work features implemented as part of the Gild environment.

Literature research does show, however, that a much larger body of research is available concerning collaborative writing tools (for example [2, 6, 11]). The joint writing of a text bears many similarities to the joint development of software, and many of the problem sets are the same.

We present a dedicated code sharing support implementation, with an interface design focused on simplicity and ease-of-use for beginners rather than extensive functionality. We found in our own class tests that group work can be more easily incorporated into a current introductory course with this tool.

The BlueJ IDE [1] forms the basis of the implementation. This offers students using BlueJ a simple way of working on projects in a co-operative fashion. In this paper, we present the background leading to various functionality requirements, the reasoning for choices made, followed by the description of a carefully considered design and implementation of such a system.

2. ELEMENTS OF GROUP WORK

Aspects of tasks of group work that lend themselves to support via software tools can be divided into three broad areas:

- sharing of artefacts
- communication
- awareness

In the interest of keeping the system simple and easy to use, we attempt to exclude all non-essential functionality, and concentrate on the tools needed for the core tasks. This allows us to design a simpler system than would otherwise be possible.

We concentrate mainly on the sharing of artefacts in the development process, and exclude much of the two other areas.

We consider sharing of the artefacts under construction – in the case of software development: source code, design documents, diagrams, and other project files – the most fundamental source of problems of the three areas named above, and thus the one that can profit most directly from additional tool support.

For a team of developers to effectively collaborate on the development of software applications, there is an obvious and central need to be able to communicate about aspects of the system under development. While communication is an important aspect of group work, there are many existing communication tools and mechanisms that students may employ. These include face-to-face meetings, instant messaging, email, SMS, social networking websites and video conferencing. Once suitable sharing mechanisms exist, it may be useful to further analyse the ways in which the sharing of artefacts and communication mechanisms may interact and be integrated.

Awareness refers to the need for the dissemination of project and process state to other team members. The availability of this information in a timely manner and relevant form allows team members to co-ordinate their activities. We consider awareness in this paper in the context of the design and development of a tool that addresses the primary aspect of supporting the sharing of software application artefacts.

3. DESIGN CONSIDERATIONS

In this section, we describe a design for a new software tool to support group work for beginners.

For all design considerations, it is important to keep in mind that simplicity of the interface is one of the most important goals. It is, in fact, a hurdle requirement: An informal survey of instructors suggests that added complexity of the tool is the most common reason why existing group work tools are not used in introductory courses. If the interface is not simple enough, it will not matter how useful the provided functionality is – the tool will not be widely used in the classroom. Therefore, we will aim at a simple, minimal design that nonetheless provides the necessary functionality to solve the main problems at hand. Whether we achieve the right balance between simplicity and functionality will determine the success of our system.

The first fundamental decision to make is the characteristic of the assumed work situation we are aiming to support: Are group members on site together regularly and are able to meet face-to-face? Then independent concurrent work is only used some of the time, while group meetings may be used for certain tasks. Or are group members physically separated (most of the time), as they are in typical distance education courses? In this case, all communication should be electronically supported.

We initially aim for the first target group: facilitating group work for on-site groups. Firstly, this is the more common scenario, and the tools might therefore fit better for a larger user base. Secondly, the tools needed in this scenario are likely to be a subset of tools needed by entirely remote groups. Tools developed for the first scenario can then be evaluated in the context of the second, and additional tool requirements can be analysed.

Assuming co-located groups leads to assumptions about tool requirements. We started the design of a group support tool with a small set of functions that can then be evaluated and extended if required. Not all activities profit equally from tool support.

For our design, we assume that groups meet for initial analysis and design tasks in person, using traditional tools (pen and paper, whiteboards) to support design discussions. Thus, we do not include, for example, shared electronic whiteboard functionality into an initial student group support tool.

As mentioned above, our tool provides IDE-integrated support for code sharing. We first describe the functionality of this support in more detail, before discussing interface issues and other considerations.

3.1 Support for code sharing

One of the fundamental tasks that should be supported by any group programming tool is the support of concurrent work on shared source code. This is commonly seen in widely used environments aimed at professional programmers. These, however, offer much more functionality than needed for a beginning learner, and with it an increase in complexity of use.

Two competing models are used in existing tools to support this: a *locking* model and a *merging* model.

The locking model assumes that the system is aware (either implicitly or through explicit user action) that a team member is currently editing a given unit of source code, and this unit is then ‘locked’ for other group members, thus preventing concurrent modification.

The merging model allows concurrent modification of units, and later attempts to merge changes in cases where the same unit was simultaneously edited by two team members. Fully automatic merging is not always possible, and manual resolution of conflicts is needed at times.

For our design, we select the merging model as the basis of source management.

The locking model has an implicit requirement of continuous connectivity of all group members. Editing off-line copies cannot be allowed without breaking the model.

For student use, it is impractical to assume always-on connectivity for home use. To make the tool useful, it is important to support students who transfer files to their home system using removable media or dial-up connections that are not continuously available.

The most widely known source sharing systems using the merging model are *CVS* [5] and *Subversion* [10]. Our design uses the same model (and our implementation is based on these systems). As in these systems, our tool has a central repository and local copies of a project for each user. It then allows the update of local copies, and the posting back of local changes into the repository.

CVS and Subversion are, however, more complex than necessary or desirable for introductory students. Our tool design supports a subset of their functionality, concentrating on check-in/check-out of new and modified files.

CVS and similar systems allow for fine-grained resource-level control. This flexibility potentially adds complexity for beginning students. Handling operations at a project-level provides a clean and simple conceptual model. That is, a commit, for instance, commits all modified files rather than requiring a subset of files to be selected.

The code management system should be integrated into the students' work environment. Setup details for the client/server connectivity should be provided to students in a single file, so that students are not required to be able to deal with many of the technical details.

Many programming tools provide graphical representations of source code and system designs. These represent a valuable artefact that also needs to be shared amongst group members.

3.2 Generating awareness

Awareness of other group member's activities can be important in group projects. A tool offering support for code sharing must provide the user with a certain minimum of awareness information. This minimum set, we believe, should include:

- Changes made to classes in the repository.
- Additions and deletions of classes in the repository.
- The time and date of each change.
- The name of the group member who made the change.
- The presence of source code in the repository which can not be automatically merged with the local version (a *conflict*).
- A revision number for each file in the repository.

Awareness information can be divided into two categories: Implicitly generated information (information that can be generated as a side effect of a developer's common actions), and explicit awareness information (information added solely for the purpose of informing team members through additional actions).

Implicit awareness information includes data about which files have been changed, when they were changed, and by which team member. The advantage of implicit information is that no additional user overhead is necessary to create this information – it is always available.

Explicit awareness information might include explicit comments left by a developer detailing the purpose and status of changes made. The advantage of explicit information is that it can provide a much greater level of detail and context, but this comes at the cost of required additional developer effort.

There are many ways to make this awareness information available to the user but they all fit in one of two categories. The information can either be *pushed* to the IDE and thus be immediately available, or the user/IDE can initiate a request and thus *pull* the information.

The obvious advantage to pushing awareness information to the IDE is that it would allow us to extend the class diagram in BlueJ to include information about the degree to which each local class is identical with its counterpart in the repository. Each class could be shown with an additional indicator signifying its state: whether it was added or removed locally or in the repository, whether it was changed locally or in the repository, whether there are any conflicts, and so on. Group members would have immediate access to up-to-date awareness information without explicit intervention.

The problem with pushing awareness information is that it requires the student to be online while working. This would present a problem for students who use a dial-up connection or moveable media to move data between home and school. It would also require teaching institutions to install, setup and maintain a purpose-build server to handle this functionality.

Pulling the awareness information from the server can be done in two ways. Either the IDE periodically polls the information or the user actively initiates a request for it.

Automatic polling would result in the presence of awareness data which would only be up-to-date after each poll. The displayed data may be out of date at any time (and even for long times, if a connection to the server cannot be made). Displaying out-of-date awareness information could very well be worse than no information at all, since it may establish a false sense of security in users and can cause the tool to behave in an unpredictable manner.

By having the user initiate a request for the awareness information and displaying a result, the fact that the information will become obsolete is made more obvious to the user. It is also easier to make the user aware if a connection to the repository cannot be established.

In the BlueJ team tools, awareness information is available through a 'status' command which displays a teamwork status dialog with a list of the resources in the project. A resource can be any type of file associated with the project.

The presence of conflicts, and thus the need to perform manual merges, is important awareness information. BlueJ notifies the user of conflicts, if any, upon performing an update operation. The user is shown a list of conflicting classes and offered a shortcut to opening them in the BlueJ editor to merge manually.

One element of explicit awareness information is supported in our tools: When committing changes to the repository the student is prompted for a comment describing the changes. Group members will later be able to view these comments by opening the project history.

The project history shows the time and date for each file change, the name of the student making the change, a version number for the file and the commit comment supplied by the student. This allows group members to keep track of each others progress.

3.3 Simplicity

As mentioned before, the underlying model and the implementation of our tool is based on CVS and Subversion, the most widely used source sharing tools. As one of the main goals is practicality – removing avoidable potential problems to adoption – this is an important factor. Many teaching institutions already have a CVS or Subversion server running, reducing the amount of system administration needed to start using the group work features of BlueJ.

Both CVS and Subversion offer more functionality than necessary for introductory projects, and we need a system that exposes only a selected subset of their functionality.

The process of adding and removing classes from the repository has been simplified. Adding or removing a class in the BlueJ project is reflected in the repository at the next commit operation. The same is true for support files, such as data files or images. Any file present in the project folder is automatically placed under control of the repository.

In addition, several large sections of existing CVS/Subversion functionality are not present in the BlueJ team support tool. This functionality includes creating code branches, tagging, version roll back (restoration of previous versions) and single file functions.

The BlueJ team support tools therefore present a limited subset of the professional version control systems, allowing us to significantly simplify the interface. This trade-off is in keeping with the spirit of the original BlueJ design, which aims to offer the most commonly needed tools, while keeping the complexity hurdle to entry as low as possible.

3.4 Interface design

A crucial element contributing to acceptance or rejection of this tool by teachers and students is the design of the user interface.

It is essential that the user interface is easy enough to understand and use that students can learn to competently employ it in their work after little introduction. Introductory programming courses often have little time to explicitly deal with group support tools, so a high level of technical sophistication in the user group cannot be assumed.

It is equally important that students have a positive perception of the benefit/overhead ratio of the tools: If the perceived immediate benefit of using the tools is not greater than the interaction overhead of its use, then students are reluctant to use it, and no benefit may be gained.

An additional constraint is catering for users who do not wish to make use of the team work tools: When group work is not intended, BlueJ's ability to support group work should ideally not add to the complexity of the interface.

In the current design, the group work tools are initially hidden, and can be enabled by selecting a checkbox in the preferences. They can also be pre-enabled globally by a teacher/administrator, should the teacher decide to use them from the beginning. When the tools are disabled, no additional interface controls are visible.

When team work tools are enabled, BlueJ shows one additional sub-menu, titled *Team*, in its *Tools* menu. This menu contains only six functions:

- *Checkout Project* – to check a project out from a repository
- *Share this project* – to place the current project into the repository
- *Update From Repository* – to update the local copy
- *Commit To Repository* – to commit changes
- *Status* – to show an up-to-date status summary of all files
- *Project History* – to show a record of all changes

The function names (e.g. *Update From Repository*) are carefully chosen to be clearer than in standard CVS, but to include standard terminology ("update", "commit"), so that students become familiar with these widely used terms.

Three additional buttons are placed into the tool bar. These buttons (Update, Commit, Status) provide shortcuts to the three most used functions from the team menu.

An interesting BlueJ-specific issue is the handling of the class diagram. Both update and commit functions include a checkbox that lets individual users include or exclude their diagram layout from the synchronisation operation. Thus, individual users can decide whether the layout should be shared between team members, or whether they like to retain an individual layout. BlueJ handles this by correctly synchronising the necessary BlueJ-specific data files.

4. MANAGING THE REPOSITORY

One of the technically most challenging aspects of using a repository based system is the setup of the server. We support and have described a simple setup scenario that serves most use cases, using standard CVS server software. This is detailed in the accompanying documentation of our system (available from the *Documentation* section of the BlueJ web site [1]). Administrators should find it straight forward to set up this server.

The user interface for students includes a group name, which is used as a subdirectory in the repository location on the server. Access control is handled by the operating system's access control mechanisms (such as Unix or Windows file access permissions).

Server information (server name, path and access protocol) can be handed to students implicitly as part of a BlueJ project that is handed out, as a file that they include into their own project, or as three pieces of information that they type in themselves.

Then, all they need to know in addition to their own account name and password is their group name. They will be prompted for this information on first use of the team support tools – that is all configuration required from students.

For the future, we are planning to provide a central public server for BlueJ users around the world. This would remove the need to have a server to make use of the team work tools for those who do not want to setup their own repository. This, however, is work in progress, and not yet available.

5. EVALUATION / EXPERIENCE

The group work functionality was first released with BlueJ version 2.2.0 in June 2007. Since then, several teachers have used the system, and we have received exclusively positive feedback.

We have used the tools ourselves for one semester in Autumn 2007 in our own class with 160 students to support teams of four to five students. The teamwork project was the last of three assignments in the first semester, starting in week 8 of the term, and lasting four weeks.

At the beginning of the team project, students received approximately 30 minutes of introduction to the team work tools. The repository model was explained, as well as the necessary BlueJ setup.

During this assignment, only two minor technical problems were reported by students, pointing to minor bugs in the implementation, which have since been fixed. No conceptual or major technical problems were reported by students.

In the student feedback survey at the end of the module, two statements were included regarding the BlueJ team work tools:

1. The team work functionality in BlueJ was useful.
2. The team work functionality in BlueJ was easy to understand.

Students were asked to score these statements on a five-point scale, from "strongly disagree" to "strongly agree". 44 students participated in the survey. Figure 1 summarises the responses to those two statements.

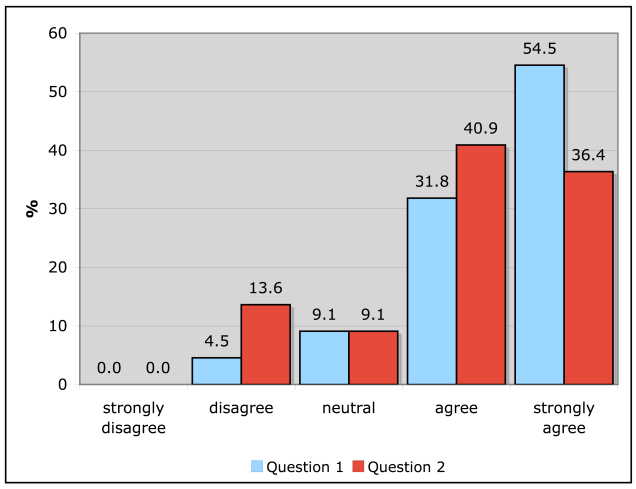

Figure 1: Summary of student feedback

The data shows that 86.3% of the students surveyed agreed or strongly agreed to the statement that the tools were useful, and 77.3% found them easy to use (agree or strongly agree).

This shows that we have largely achieved the goal of making group work functionality easily accessible to most students.

6. CONCLUSION AND FUTURE WORK

The group work support in BlueJ is available now and has been since June 2007. It provides easily learnable and accessible tools that effectively help student groups manage their team work projects. After initial analysis, we decided to concentrate on source code sharing functionality and low level awareness information, excluding high level awareness and explicit communication facilities. This has served well in keeping the user interface small and made it useable by almost all students after minimal introduction.

The current implementation is based on CVS, and exclusively supports CVS repositories. An implementation supporting Subversion repositories is currently in development, and will be included with the next release of BlueJ. The user interface will not be affected.

The largest hurdle we currently see is the installation and setup of the server software. While universities generally have technical support staff who are familiar with repository servers and can routinely support these systems, this is not always the case for smaller institutions, such as secondary schools. To alleviate this problem, we are currently investigating provision of a central repository server, publicly available to all BlueJ users, and integrated into BlueJ. This should remove the need to set up a custom server, and should make the creation of new shared projects almost as easy as creating an individual, local project.

We believe that group work is an essential skill that needs to be practiced by all future software professionals, and we believe that these tools make this goal easier than it previously was.

7. REFERENCES

[1] BlueJ. BlueJ - The Interactive Java Environment. http://www.bluej.org, accessed Jan 2008.

[2] Carratto, T. Studies of Computer Supported Collaborative Writing. Implications for System design. in Blay-Fornarino, M. ed. *Cooperative systems design : a challenge of the mobility age*, IOS Press, Amsterdam ; Washington, DC, 2002, 139 - 154.

[3] Chase, J.D. and Okie, E.G. Combining cooperative learning and peer instruction in introductory computer science *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*, ACM Press, Austin, Texas, United States, 2000.

[4] Čubranić, D. and Storey, M.A.D. Collaboration support for novice team programming *Proceedings of the 2005 international ACM SIGGROUP conference on Supporting group work*, ACM Press, Sanibel Island, Florida, USA, 2005.

[5] CVS. CVS: Concurrent Versions System. http://www.nongnu.org/cvs/, accessed January 2008.

[6] Dourish, P. and Bellotti, V., Awareness and Coordination in Shared Workspaces. in *CSCW'92*, (Toronto, 1992), 107 - 114.

[7] Ellis, W., Ratcliffe, M.B. and Thomasson, B., Promoting fairer grading of group based assessment using collaborative IT Tools. in *&th International Computer Assisted Assessment (CAA) Conference*, (Loughborough University, UK, 2003).

[8] Last, M.Z., Daniels, M., Hause, M.L. and Woodroffe, M.R. Learning from students: continuous improvement in international collaboration *Proceedings of the 7th annual conference on Innovation and technology in computer science education*, ACM Press, Aarhus, Denmark, 2002.

[9] LeJeune, N. Critical components for successful collaborative learning in CS1. *J. Comput. Small Coll.*, *19* (1). 275-285.

[10] Subversion. Subversion. http://subversion.tigris.org/, accessed January 2008.

[11] Urnes, T. and Nejabi, R. Tools for Implementing Groupware: Survey and Evaluation York University, 1994.