

An Infrastructure for Teaching Skills for Group Decision Making and Problem Solving in Programming Projects ¹

Amer Diwan
Computer Science
University of Colorado
Boulder, CO 80309
diwan@cs.colorado.edu

William M. Waite
Electrical Engineering
University of Colorado
Boulder, CO 80309
waite@cs.colorado.edu

Michele H. Jackson
Communication
University of Colorado
Boulder, CO 80309
jackson@colorado.edu

Abstract

In industry, programmers work in groups to design and implement substantial pieces of software. In contrast, most programs that students write in classes are toy programs involving little or no group work. To address this discrepancy, we have developed a software infrastructure that aims to teach group work skills to students in computer science courses and also enables students to tackle larger and more significant projects. We are in the process of deploying this infrastructure in a three course sequence at the University of Colorado: Data Structures—Programming Languages—Compiler Construction.

1 Introduction

Successful engineers must be able to recognize and fill gaps in their education, work effectively in groups, and use past experience to predict future performance. In order to prepare students to be successful engineers we must teach them the set of scientific knowledge underpinning their particular discipline. In addition, we must teach them to recognize holes in their knowledge and fill those holes, to work with others in both knowledge acquisition and artifact creation, and to plan their activities based on an accessible history of previous projects.

Translating these general educational objectives into specific classroom environments is a serious challenge. First, limitations of available design tools tend to constrain instructors to the use of relatively well-defined,

¹This work is supported by NSF grant CCR-0086255. Any opinions, findings, and conclusions or recommendations expressed in this material are the authors' and do not necessarily reflect those of the sponsors.

simple problems that bear little resemblance the problems that are faced by practicing engineers. Second, engineering curricula typically do not include courses specifically intended to develop skills needed for group work. Since group projects are typically only one of the many components of existing courses, these projects are inadequate for teaching students the necessary group-work skills.

Indeed, data from a compiler construction course at the University of Colorado [7] suggests that students actively seek strategies to avoid genuine group work, relying instead on dividing up the task or placing the whole responsibility on a minority of the group. When we extended the class to incorporate instruction in group interaction, students reported feeling overwhelmed at having to learn what they saw as two completely unrelated skill sets. This evidence suggests that one cannot simply assign students to work in groups and expect group skills to develop, regardless of whether or not instruction is provided. One needs to provide them with the opportunity to work on non-trivial problems in a complex environment. Fischer *et al.* [11, 8] report a similar experience when they tried to get students in a class to generate new ideas as a group.

In this work, we identify three skills necessary for group work and describe an infrastructure that helps to teach these skills to students over the course of three semesters. The group-work skills we specifically target are: (i) an appreciation for *interdependence* and the ability to recognize when a task is dependent on the efforts and accomplishments of others; (ii) the ability to consider and argue for or against different viewpoints in a constructive manner [12] (*negotiation*); and (iii) a familiarity with structured, systematic decision-making procedures (*group problem solving and creating*).

The key idea behind our approach is to involve students in realistic group projects beginning with the introductory courses but to *use an infrastructure to selectively hide the complexity from the students*. As students progress to more advanced courses, the infrastructure

reveals more of the complexity of group work to students, until finally, students are involved in realistic projects involving significant group work. Our implementation, called the *repository*, has four main parts: (i) a web interface which both the students and the instructors use to access the system, (ii) the *sentry* that manages the complexity that is exposed to the students, (iii) a database that stores all relevant data (such as student records and assignments), and (iv) a collection of tools that allow students to undertake larger projects than would normally be possible in a classroom setting.

The remainder of this paper is organized as follows. Section 2 describes the repository in greater detail. Section 3 summarizes related work in the area and Section 4 concludes.

2 Infrastructure and Strategy

The *repository* is the heart of our system (Figure 1). The repository maintains a database which has all information about students and also contains student and instructor code modules. The *Sentry* module enforces the policies for classes and assignments. The *analysis*, *abstraction*, and *planning and reflection* tools assist students in designing, writing, and reasoning about programs (Section 2.6). We wrote all web and database code in Apple's WebObjects/EnterpriseObjects frameworks [6]. We first describe the repository from the perspective of students and the instructors (Section 2.1 and 2.2) and then describe how we will use it for teaching group-work skills to students (Sections 2.3, 2.4, and 2.5). Section 2.6 gives examples of some of the tools that we will use with the repository. Note that the repository is still under development and the current capabilities of the repository are a subset of what we describe in this paper.

2.1 The repository, from a student's perspective

Students access the repository through a web browser. Once they have logged in (using their username, password, and class), they pick an architectural environment (e.g., X86/Linux). The environment determines the format of any binaries that the repository may return to the student.

Once the student has picked an architecture, she has the following options:

1. Upload new files or new versions of existing files. In this way, students are effectively using the repository as a configuration management system, e.g., cvs. The repository simply provides a high-level interface to an existing version control system.
2. Refresh versions of group member's files. If a student updates a file, her group members do not get

the new version automatically: they must explicitly refresh that file to bring it up to date. The benefit of this approach is that if a student uploads a buggy version of a file that, for instance, does not even compile, the group members are not affected and they can continue to use the older version. The repository does, however, inform students when a group member updates a file.

3. Select one or more files and perform an *action* on them. The selected files may include not just the student's own files but also files belonging to group members and files that the instructor has made available for the assignment (*instructor's files*). An action may be simple, such as compilation or viewing the contents of files. Or actions may be more sophisticated, such as program slicing [18]. When the repository completes an action, it returns the action's output and errors to the student. The architectural environment determines what actions are available. The *sentry* determines what actions can be applied to what files. For example, a student may be allowed to compile against files checked in by the instructor but may not be allowed to view them.
4. Post or respond to a bug report. If a group member has registered a bug in a student's code, the student is notified. The student must either fix the bug or determine (in negotiation with other members of the group) that the bug is in someone else's code.

2.2 The repository, from an instructor's perspective

The instructor also uses a web browser to access the repository. Once an instructor has logged in, she has the following options:

1. Add new students to the class or modify existing student information.
2. Create new programming assignments or modify existing assignments. Each assignment has three main parts: (i) a description of the assignment including deadlines; (ii) a collection of files to be used in the assignment (*instructor's files*); and (iii) a set of policies for the assignment. If an assignment includes files, then these files are available to the students when attempting this assignment. The allowed operations on these files are determined by the policies for the assignment. There are three sets of policy decisions that an instructor must make for each assignment:
 - Are students allowed to view the files that are part of the assignment or can they only use them as input to *binary* actions, such as compilation (see below for description of actions);

| | | | |
|--------------------------|----------------|-------------------|-------------------------------|
| Web based user interface | | | |
| Sentry | | | |
| Database access | Analysis tools | Abstraction tools | Planning and reflection tools |
| WO/EO | | | |

Figure 1: Architecture of the repository

- Are students allowed to view the files of other group members or can they only use them as input to *binary* actions?
 - Is there an expiry date on the files provided with the assignment or are these files available for the entire duration of the assignment?
3. Group students for existing assignments. The grouping may be manual or automatic (e.g., randomly selected groups of a given size) [17]. We are also considering mechanisms where the automatic grouping takes into account the past performance and background of students.
 4. Create new architectural environments (e.g., X86/Linux). Each architectural environment consists of a number of *actions*. An action is a command that can be executed on a machine of that architecture. For example, an action might be “gcc -v -g -o” to compile with gcc. Each action can take one or more file names. The first file name always specifies a location for the output of the action. Subsequent files are inputs to the action. A *binary* action does not return the source of its input file and therefore students can use it even on files whose source they are not allowed to view.
 5. Create or modify new machines for an architectural environment. A machine is simply the address of a real machine that can execute the actions for its architectural environment. The repository executes an action by copying files over to a temporary directory on a machine, executing the action, and copying the files back. To ensure security and efficiency, the repository uses ssh [2] to execute commands and rsync [4] (on top of ssh) to transfer files. When defining a machine, an instructor must indicate what temporary directory to use on the machine and the ssh identity to use.
 6. Grade students and groups based on the data in the repository. The repository provides detailed information about student performance (such as timeliness of a student’s response to bug in her code) which the instructor can use for assigning grades.

2.3 Introductory course: Teaching an appreciation for interdependence

In this course, students work in a “group” with the instructor to implement a significant project with multiple stages. The instructor provides the code modules for the entire assignment; students replace these modules one at a time over several stages. Students are not allowed to view the source code for the instructor’s modules but can compile their own modules against them.

The key advantages of this organization is that students get experience in being being *interdependent* on someone else for their project. More specifically,

- Students are exposed to the idea of being responsible for only part of the project at any given time. They get used to debugging their code while it is interacting with someone else’s code that they cannot even see.
- Since students are grouped with the instructor they do not have to worry about some of the most difficult aspects of collaboration, such as how to deal with other group members of different skill and commitment levels.

2.4 Intermediate class: Teaching negotiation skills

In this course, students work on a group project with other students. However, the instructor provides a detailed design. The students negotiate amongst themselves to subdivide the project. The instructor provides code for all the modules in the project (though the instructor’s code may not have all the functionality that the student’s code eventually needs to have). Also, instructor’s modules may be compiled against but cannot be viewed in source form or downloaded. The assignment is also set up so that students can compile against group member files but cannot view them using our system. Group members are, however, allowed to look at each other’s code for code walkthroughs and such. This is similar to real-world situations where it is not feasible for an individual programmer to have a deep knowledge of the entire program which may have millions of lines of code. The instructor may require students to do some

planning and reflection (e.g., keep track of the time they spend on each aspect of the project) and to use this information to plan for subsequent stages of the project [7].

Students interact with each other by checking in files, by registering bug reports against each other's code, and by resolving bugs.

To get the maximum credit for this assignment, the group's code should work together as a whole. However, if a student has finished her share of the code and the code works when linked with the instructor's code then the student can still get the majority of the credit for the assignment.

This project focuses on teaching students how to negotiate in a constructive manner based on their technical knowledge. More specifically the benefits for this project in teaching group-work skills are:

- The initial breaking down of the project and the bug reporting and fixing mechanisms give students experience in negotiation based on technical information. For example, Student A may report a potential bug in Student B's code. Student B must either fix the bug or must determine that the bug must be in Student A's code which passes incorrect parameters to Student B's code. Students A and B need to negotiate based on technical knowledge of the program and their respective codes in order to determine the source(s) of the bug.
- Since students are rewarded for finishing the project as a group, this project continues training students in how to deal with interdependence.
- It provides a low-risk environment for teaching group work since students are not overly penalized if their group members do not deliver their share of the work.

The repository also provides us with additional knobs that we can use to tune the project. For example, by setting an expiry date on the instructor's modules, the instructor can reward students that start early on the project. As another example, the instructor can choose to reward students that respond to bug reports in a timely fashion.

2.5 Advanced class: Teaching group problem solving skills

The organization of this project is similar to the project in Section 2.4. However, here, students are also responsible for *designing* the project based on loose guidelines provided by the instructor. In this project, the repository primarily serves as a configuration management system, a communication mechanism (for tracking bug reports), and a set of tools that the students can use

as they see fit. Students are required to use the planning and reflection tools during all stages of the project so that the negotiation, design, and planning processes are well documented (see Section 2.6 for an example of these tools). In our prior experience we have found that this documentation can yield valuable insights about the student's approach.

This project will give students experience in how to solve problems and create software as a group. Since the tasks in this project are a superset of the earlier two projects, this project further reinforces the skills needed for interdependence and negotiation. The planning and reflection tools for recording their interactions provide valuable feedback to the students and instructors on how well the group work is proceeding and what can be done to improve it.

Comparing the project in this section with the projects in Sections 2.3 and 2.4 we note that as the projects become more complex our infrastructure becomes increasingly invisible. In the advanced project described in this section, the infrastructure just takes the form of a uniform interface on a number of standard programming tools (compilers, bug reporting systems, and configuration management system). Thus, it is worth emphasizing that our system is not a group work tool: it is a tool for *teaching* group-work skills.

2.6 Tools

The previous section alluded to tools that students can invoke via actions. In this section we elaborate more on the kinds of tools that we envision having as part of the repository. The action mechanism allows an instructor to incorporate tools of their own with ease.

Analysis tools allow students to analyze a small program or routine in order to better understand both what it is doing and how it is constructed. The analysis may be based on either compiler technology or some kind of profiling. The role of these tools is to help students to deal with the complexity of interacting with code that someone else has written. Some tools that we feel will be particularly useful here are: compilers, data structure visualizers [16], program slicers [18], automatic invariant discovery tools [10], source level profilers [14], and shape analysis [9, 13].

Abstraction tools accept high-level problem descriptions and create robust implementations according to standard solution patterns. They allow students to concentrate on principles rather than details, and to tackle more realistic problems within the time frame of a course. More specifically, *Eli* is one such tool that can be invoked via our system [15].

Planning and reflection tools capture information

and insights about a project as it is being carried out and make them available as a basis for planning future work. They will be used to make students' behavior visible and allow them to reflect on both the technical and social aspects of completed assignments. One example of such a system is the *Personal Engineering Process* tool [3] for planning and reflection.

3 Related Work

While there has been much work on tools for group work (e.g., SourceForge [5]), we are not aware of any tool that is specifically designed for *teaching* group-work skills to computer science undergraduates.

There has also been significant work on methodologies for group work. For example, Extreme Programming [1], a methodology for various aspects of software development defines how a group (which includes not just pairs or larger groups of programmers but also customers) works together in developing a software system. Our infrastructure simply gives mechanisms to instructors for teaching group-work skills; the instructor can pick appropriate policies for enforcing particular programming or group-work methodologies.

4 Summary

We have developed a tool to teach group-work skills to students in computer science courses. We describe how to use this tool in a three course sequence of computer science courses. As the courses get more advanced, our system reveals more and more of the complexity of group work to the students. In this way, our tool allows students to slowly build up their group-work skills over a period of three courses rather than exposing them to the skills all at once.

References

- [1] eXtreme Programming: A Gentle Introduction. <http://www.extremeprogramming.org>.
- [2] Openssh. <http://www.openssh.org>.
- [3] *Personal Engineering Process*. (<http://deming.colorado.edu/ecen4553/peptoc.html>).
- [4] rsync. <http://www.rsync.org>.
- [5] SourceForge. <http://www.sourceforge.com>.
- [6] Webobjects developer documentation. <http://developer.apple.com/techpubs/webobjects>.
- [7] Carter, L. R., and Waite, W. M. Problem solving skills. In *Frontiers in Education* (2000).
- [8] dePaula, R., Fischer, G., and Ostwald, J. Courses as seeds: Expectations and realities. In *Proceedings of The European Conference on Computer-Supported Collaborative Learning* (Maastricht, Netherlands, March 2001), pp. 494–501.
- [9] Deutsch, A. Interprocedural may-alias analysis for pointers—beyond k-limiting. In *Proceedings of the SIGPLAN Conference on Programming Languages Design and Implementation* (San Francisco, CA, June 1992), pp. 230–241.
- [10] Ernst, M. D., Czeisler, A., Griswold, W. G., and Notkin, D. Quickly detecting relevant program invariants. In *Proceedings of the 22nd International Conference on Software Engineering (ICSE 2000)* (Limerick, Ireland, June 2000).
- [11] Fischer, G. Creating the university of the 21st century: cultural change and risk taking. <http://www.cs.colorado.edu/~gerhard/reports/cu-risktaking1998.pdf>, October 1998.
- [12] Folger, J. P., Poole, M. S., and Stutman, R. K. *Working through conflict: strategies for relationships, groups, and organizations*. Longman, 1997.
- [13] Ghiya, R., and Hendren, L. J. Is it a Tree, a DAG, or a Cyclic Graph? A shape analysis for heap-directed pointers in C. In *Proceedings of the 23rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (1996), pp. 1–15.
- [14] Graham, S. L., Kessler, P. B., and McKusick, M. K. gprof: A call graph execution profiler. In *Proceedings of the SIGPLAN '82 Symposium on Compiler Construction* (1982), pp. 120–166. Also available as SIGPLAN Notices, Volume 17, No. 6.
- [15] Gray, R. W., Heuring, V. P., Levi, S. P., Sloane, A. M., and Waite, W. M. Eli: A complete, flexible compiler construction system. *Commun. ACM* 35, 2 (February 1992), 121–131. <http://www.cs.colorado.edu/~eliuser>.
- [16] Korn, J., and Appel, A. Tree-based visualization of data structures. In *IEEE information visualization* (October 1998), pp. 11–18.
- [17] Redmond, M. A. A computer program to aid assignment of student project groups. In *Special interest group on computer science education (SIGCSE)* (Charlotte, NC, February 2001), pp. 134–138.
- [18] Weiser, M. Program slicing. *IEEE Transactions on software engineering* 10, 4 (1984), 352–357.