In *Proceedings of the Software Engineering: Education & Practice Conference*, Dunedin, New Zealand, January 1998. IEEE Computer Society Press.

Group-Based Software Engineering in an Introductory Computer Science Course

Craig E. Wills*
Computer Science Department
Worcester Polytechnic Institute
Worcester, MA 01609 USA
cew@cs.wpi.edu

Abstract

We have explored the use of peer learning in a large, introductory data structures course within our computer science curriculum. The principal peer learning activities are group programming projects where each student group gains first-hand experience in working with others and putting into practice the basic concepts of software design and engineering.

This paper describes the use of group programming projects in the course and provides our experience with them in three instances of the course over a four year period. Overall, we have found the introduction of group programming projects in this large introductory course to be beneficial to students in gaining first-hand experience with a large programming project, which both motivates the need for and provides initial experience with good software engineering principles.

1 Introduction

As in many computer science curricula, our department devotes a course to software design and engineering. This course is typically taken by students late in their 2nd year or during their 3rd year of study. Some introductory software engineering concepts such as design, abstraction and testing are introduced as part of our CS2 [1] course primarily devoted to data structures. With the study of data structures, students begin to have the programming tools necessary to construct large scale programs, which require a basic understanding of software engineering principles. However, it is difficult to design programming projects for a single student that demonstrate the need for real software engineering principles.

A solution to this difficulty is to introduce group programming projects, but there are a number of potential problems that arise from doing so at the introductory level:

- class sizes are larger than typically found for an upper-level software engineering class creating less teacher/student interaction and much potential administrative overhead for managing groups,
- students are both trying to understand data structures and how to use them in designing large projects,
- introductory students are less likely to have experience working in groups, and
- students have received less formal instruction on software engineering principles before needing to put them into practice.

Despite these potential problems we have introduced the use of peer learning in the form of group programming projects in our CS2 course. Not only were we concerned with providing students with more realistic software projects, but we wanted to increase student interaction in a large introductory class and to instill in students the need to take responsibility for their learning and that of those around them. In support of this approach we have introduced the use of trained, upper-level under-class undergraduate peer learning assistants (PLAs) to help facilitate student group interaction and developed software to minimize the administrative overhead of handling many groups. The PLAs are consultants to the group to help facilitate group interaction and support group and individual questions.

*Currently on leave at Victoria University of Wellington, New Zealand.

Through the use of peer learning assistants and the creation of software to minimize the administrative overhead of groups we have been able to concentrate on making the course a quality educational experience. This paper describes the use of group programming projects in the course and provides our experience with them in three instances of the course over a four year period.

2 Background

Worcester Polytechnic Institute is a private university with approximately 2800 undergraduate students and 300 computer science majors. The academic calendar at WPI consists of four 7-week terms with students typically taking three courses each term. Each course typically meets four times each week with an additional meeting for courses with laboratories. Students generally take courses for four years to complete a Bachelor of Science degree.

CS2005 is the recommended gateway course for students wishing to take upper-level computer science courses and is taken after an introductory programming class. It is intended to accept students who have had a single term of computer programming and to develop skills in the design, implementation and analysis of basic data structures—similar to the traditional CS2 course [2]. Students are also expected to gain initial experience with programming-in-the-large and software engineering principles. The course currently uses the C++ programming language with programming and laboratory assignments done on workstations running a version of the Unix Operating System. The audience for this course is diverse—including not only computer science majors, but large numbers of non-majors, particularly from departments such as Electrical and Computer Engineering, Math and Management.

Previous work relates to our approach of using peer learning in this course. Collaboration is typically introduced in upper-level software engineering courses, and experience has been reported at this level [3, 4]. Group work has been done in many forms within other courses across the computer science curriculum [5, 6, 7]. Other work has examined the use of undergraduate teaching assistants [8].

3 Approach

The approach we have taken is to motivate students to accept more responsibility for themselves through use of peer learning as described in [9]. Students take part in small group activities in class and in programming projects outside of class. Apart from being a natural approach for software design and development, it is appealing for its potential to draw students together and help them learn from each other. The goals in using this approach are:

- to increase the number of contacts among students in the learning process, which we believe will increase the accountability of students for their work and others,
- to assign more realistic programming projects that are large enough to require students to consider software design and engineering,
- that student interaction will increase and become a larger part of the student learning process than traditional student/instructor, student/TA interaction, and
- to employ support mechanisms that will promote this course organization without overwhelming faculty resources.

The issues are how to bring collaborative learning to the classroom without overwhelming faculty in administrative overhead, and so that students not only accept it, but "buy in" to it. We have taken a two-pronged approach: 1) use peer learning assistants (PLAs) to help facilitate the peer learning process; and 2) use software not only as part of teaching the course, but to lessen the overhead of administering such a course, thus making the course viable for an individual faculty member to teach. This approach affects all facets of the course, which are detailed in the following sections.

4 Course Structure

The weekly course structure is four 50-minute class meetings with all students and one 50-minute laboratory of about 20 students. The class meetings are led by a faculty member. The labs are led by graduate teaching assistants (TAs) for the course. Peer learning assistants are upper-level undergraduate students who facilitate group work among students.

4.1 In-Class Activities

The class meetings involve lecture material by the instructor mixed with informal active learning activi-

ties and discussion [10]. A key component of the in-class activities is organized group work, with at least one of these activities each non-exam week.

A common activity is a "group quiz" where students group themselves based on seating proximity. The quiz may require the students to consider different design possibilities or apply concepts learned in class to a different problem. While the students work and discuss, the faculty member circulates through the classroom answering questions and facilitating discussion. After the groups have worked on the exercise for most of the meeting time, the class spends time discussing problems encountered. The groups then have time to review their work.

In-class group exercises allow students to participate in active learning activities in a comfortable environment while maintaining direct involvement with the instructor. We believe these type of exercises are important for students to understand the value of collaborative learning in a setting that involves the instructor. They also allow the faculty member to better understand where students are experiencing difficulties.

4.2 Laboratory

The course includes a weekly closed laboratory where each student has access to a workstation. The labs are led by a TA and assisted by the PLAs. Students learn about and obtain experience with software tools useful for debugging and building software projects. The labs are also used to study a particular topic, such as the comparative timing of different sorting procedures. Learning rather than evaluation is stressed in the labs so students are encouraged to work together and with the TA.

Lab periods are also used for peer learning activities related to the group programming projects. Student groups have time to meet with each other and their PLA to discuss the current assignment and how their group is proceeding on the assignment.

A typical set of seven labs for the course is given below including topics and tools introduced in each.

Week 1 Students register themselves online in the class database. Students review Unix, editors and compilers if they are not familiar with their use.

Week 2 Students meet in their programming project groups for the first time. This lab is led by a PLA who leads students through team building exercises including selection of a group name and a code review.

Week 3 Students learn about the Unix tool *Make* [11] for building multiple software modules together.

Week 4 Students review problems with pointers by testing and debugging a program using a debugger.

Week 5 Students learn about a number of scripting tools available in Unix and how these tools are combined in scripts. They learn that tasks can be accomplished by other means than creating a new program.

Week 6 Students experience the use of recursion through a simple recursive drawing language.

Week 7 Students compare the performance of different sorting algorithms on various sizes of input. Performance is plotted using the *gnuplot* plotting tool.

5 Group Programming Projects

The most significant peer learning activities are group projects done outside of class, which focus on larger design and programming problems. There are three such assignments during the course of the term following an initial assignment done on an individual basis that leads into the group work. The projects are done in pre-assigned groups of 4-5 students. Students in these groups share a common lab time with compatibility of out-of-class schedules. Technical and interpersonal skills are also taken into account when forming groups. Each group is led by a PLA to help facilitate group activity. All students on the project receive the same grade unless individual assessments indicate otherwise.

The first project in the course takes one week and is designed to help students review material from their first programming course (typically in C) and help students "get up to speed" with the language used in the course. A typical assignment used for this task relates to mathematics and involves representation and manipulation of polynomials. Students create an array implementation and implement symbolic integration and differentiation, along with evaluation of the polynomials. This project begins as an individual assignment, but during this time the students are formed into groups. As the first group activity they perform a code review of each other's work to obtain peer feedback on what they have done and to learn from examining what others have done. They also get to know each other and choose a group name.

The second activity is given a little over a week into the course after the students have been exposed to stacks and queues. This project is given as a group assignment. The purpose of the project is for students to learn about data abstraction and to complete a project that is broken into pieces. One such assignment used in the course is adds breadth to their knowledge of computer science by exploring the problem of job scheduling in operating systems. The students create a driver program to simulate the arrival of jobs into their system with specific start times and duration. In addition, each student creates routines to manage these jobs based on a different job scheduling policy. Each policy uses either a queue or stack to maintain the jobs.

Details of the assignment are covered in a peer group meeting outside of class and laboratory time. If possible, this meeting includes the PLA. Students discuss the overall organization of the project and how it can be divided for work among the group members. With the help of the PLA, students take responsibility for pieces of the project based on individual styles and interests.

After distributing work among individual group members, the students work on their portions and communicate (often electronically) with each other and their PLA on progress and problems. Group work is facilitated with a electronic mail alias for each group and a group-specific directory where group members can do their work and share it with the group. The students have access to each other, the PLA, teaching assistants and faculty for individual or group assistance. The students' work culminates by building the individual modules together into a working whole and testing them on various data. Students turn in their group contributions electronically along with an individual evaluation of each group member's contribution to the whole.

The next major project in one of the course offerings was to build an airline reservations system. This project uses more of the techniques the students are learning (linked lists, hashing and searching) and involves aspects of databases and heuristics for storing information and routing passengers. As before, the groups break up the project among the group members who must work together to build the entire system. The project is significant in scope and tests students' abilities to design, code and organize it.

The final group project involves trees and recursion. In this project, the students are given a data structure representation for a tree and create a set of routines, most of them recursive, to build and manipulate it.

5.1 Project Lifecycle

Figure 1 summarizes the lifecycle of a group project and shows the roles of the instructor, TA and PLA. The instructor of the course is the overall manager for the activities in the course. He or she creates the group assignment and is ultimately responsible for questions or problems. As shown, the design for the project is done by the entire group and broken into parts for individual work. Once individual pieces are coded and tested, the parts are built together and tested. The PLA helps facilitate group work and is available for technical questions during all phases of the project.

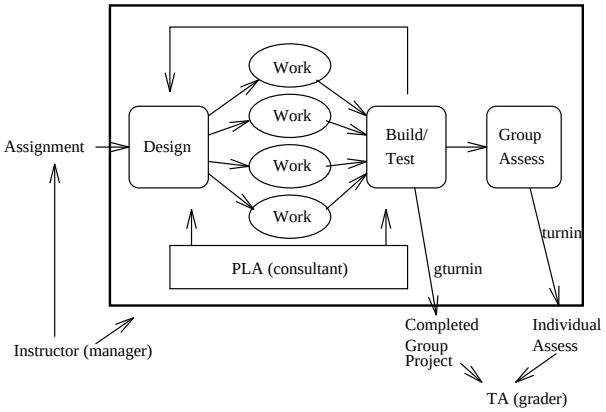

Figure 1: Project Lifecycle for a Group Project

Once the project is complete, the students use a software tool *gturnin* to electronically turn in source code and tests of the completed group project. Group members also discuss how the project went; after which each member uses the software tool *turnin* to submit his/her assessment of how the group worked on the project along with individual assessments on how each member contributed to the project. The completed group projects are graded by the TAs.

5.2 Project Assessment

Group projects are graded as a group with this grade used as a base for individual grades. At the end of a project, each group member turns in an individual assessment of all members contributions to the group for the project. These assessments are in the form of a multiplier, which are multiplied by the group grade to yield individual grades. Thus a situation where all group members contributed equally is indicated by all multipliers of 1.0. Multipliers of greater than 1.0 are allowed, but generally ignored unless the group as whole has done poorly, but one or two individuals has

done more work. In practice, 90% or so of the individual project grades are the same as the group project grade.

In terms of group accountability for individual learning, there are three exams given in the course emphasizing important material. Particular questions on the exams relate directly to group activities and are noted as such. Each student group is then given a group score for the exam based on individual group members' performances on the group questions. Groups whose individual members do well receive bonus points. This aspect of the course emphasizes the accountability of students to not just complete the work, but for all group members to understand it.

6 Software

Not only do we use software to aid in teaching the course, but a key aspect to reducing the administrative overhead is the development of software to help in managing the course [12]. Although this software does not relate directly to the student learning process, it is necessary to maintain records on grades and group activities. We had some prior experience with electronic submission and grading of projects. We built on this experience to make the administration of group work manageable for large classes. Without such software we believe the administration of such a large, collaborative-based class would be difficult. The software we developed is compatible with Unix environments. Its major components are described as follows.

- The most important piece of software is an electronic survey program. We use it to collect information for assigning students to groups and for gathering feedback at the beginning and end of class for assessment and evaluation.

- Once student information is collected another script is used to semi-automatically group students based on the laboratory section, out-of-class schedules, and self-assessed technical and interpersonal skills.

- Students turned in projects as a group using *gturnin*. Students also submitted individual assessments of the group performance using *turnin*.

7 Variations on Our Group Project Model

Three offerings of the course using this basic model have been taught by the same faculty member at the same time during the academic year. Each of these courses has its own variations, but the basic model has been constant. Variations in the three courses are summarized in Table 1. More complete results are in [13].

Table 1: Offerings of Course

Year	Class Size	# PLAs	Group Org.	Lang.
1993	128	6	random	C
1994	130	2	self-assess	C
1996	187	6	self-assess	C++

The initial offering of the course was in 1993 with 128 students. Six PLAs were selected from student applications and these PLAs were trained for their role in the course (PLAs are paid for their time devoted to the course). Students in the class were organized into groups based first on the lab section they attended (labs contain 20-22 students). In this way students were assured of finding their group members during lab time. In addition, students were surveyed on their out-of-class availability for meetings and organized into groups of 4-5 accordingly. Other than lab and out-of-class time availability, the students were grouped randomly.

One result of our initial offering was that the randomly formed groups left a few groups lacking interpersonal or technical leadership. These groups often had problems in successfully working on projects, which led to group conflict and often a visit to the faculty member's office by one or more group members. This result led us to reexamine how groups were formed. Some students suggested choosing their own groups, but in such a large class with students not knowing each other well this approach did not appear easy to implement nor equitable to all students.

Based on our observation that successful groups had technical and interpersonal leadership, we modified the way in which groups were formed in the 1994 class (using the same approach in 1996). The results of five survey questions (given as part of a survey at the beginning of the class) were used to compute a "goodness" factor for each student based on the student's own self-assessment. Once each student's goodness factor is determined then groups were again formed

based on a common lab time and where possible common out-of-class meeting times. However, an overriding concern was to equalize the average goodness rating for each group.

In 1996, the course population increased dramatically to 187 students. This increase was not linked to the approach used in the course, but rather based on a large increase in majors and general interest in computing. We used our same model, but did increase the number of PLAs to six (although a higher student to PLA ratio than in 1993). Other than class size, the only other significant change was the use of the C++ programming language (no overloading or inheritance) in the course versus the previous use of C.

8 Results

We obtained information and feedback from the students in a number of ways: a survey at the beginning of the course (numeric responses), a survey at the end of the course (numeric responses), a survey at the end of the course (written responses) and a standard WPI course evaluation sheet. Quantitative student perceptions on aspects of each course were gathered based on a 1-4 numeric scale where one is strongly disagree and four is strongly agree with a given statement. Averages for questions selected from the final survey common to each course are shown in Table 2.

The results show few dramatic changes between the three versions, but overall the best results occurred in 1993. This result is predictable given that the 1993 version had the best student to PLA ratio along with the initial enthusiasm and determination to make a new approach work. The 1994 version went fine, but the drop-off in some of the indicators is likely the result of fewer PLAs and a little bit of a let down in instructor enthusiasm. The 1996 version was similar with the lower student to PLA ratio helping the ratings, but the large number of students hurting. Similar results for student satisfaction were found from the standard WPI course evaluation form.

An interesting result from all three versions was the perceived amount learned versus course grade because of the group projects. In all cases, the amount learned was the same or higher than expected for individual projects, but on the other hand the perceived grade was always lower because of the projects. Our experience as faculty indicates the course grading was not changed because of the group projects. We believe the perception comes from better students who feel they are dragged down by having to work with less capable students in the class. However, it is precisely this

Table 2: Average Course Survey Results(1=SD,2=D,3=A,4=SA)

	1993	1994	1996
I learned a lot in this course	3.4	3.3	3.2
I liked working on projects with other students.	2.9	2.9	2.7
The people in my group worked well together.	2.5	2.5	2.7
My PLA was very helpful to the group.	2.7	2.0	1.9
I enjoyed this class.	3.2	3.0	3.0
I worked hard in this class.	3.3	3.1	3.2
I feel I learned (1=much less, 2=less, 3=about the same, 4=more, 5=much more) because of working with others in group projects rather than working on individual projects.	3.3	3.2	3.0
I feel my course grade will be (1=much lower, 2=lower, 3=about the same, 4=higher, 5=much higher) because of working with others in group projects rather than working on individual projects.	2.9	2.9	2.7
How do you rate your overall level of satisfaction with this course? 1=very disappointed, 2=somewhat disappointed, 3=somewhat satisfied, 4=very satisfied.	3.1	2.9	3.1

interaction we are encouraging with groups where in the best case all students learn more.

There is still a problem faced by groups of how to deal with weaker or less motivated students whose failure to meet group deadlines potentially impacts the success of the entire group. The strong interdependence of group programming projects can cause tension in the group when this situation occurs and is from our perspective consistently the most difficult aspect of using group programming projects. One potential solution is allow students to be "fired" from groups, but this may have the negative effect of removing a student from a group who needs help from others the most. We have not used this approach, but in rare instances have disbanded groups where it seems to be best for the students.

From the perspective of software engineering, we have found that the programming projects point out (sometimes painfully as the project deadline looms) to the students the need for good software design and testing. Particularly for the more complex group projects, groups with problems had a poor design or did testing only after integrating all of the modules. Over the course of using this approach, we have improved on our design of projects for easier decomposition into individual pieces of work within the overall group. However, students still have problems with abstraction and integration as they also try to understand the basic concepts of data structures, searching and sorting.

9 Summary

Overall, we have found the introduction of group programming projects in this large introductory course to be beneficial to students in gaining first-hand experience with a large programming project. This approach has been beneficial to students, PLAs, TAs and faculty. The students are able to legitimately work with peers to solve more realistic problems and learn the material. Peers are more able to provide a student perspective in answering questions. PLAs have benefited from the approach by involving themselves in the teaching process and having a much better understanding of group dynamics. The big benefit for TAs and faculty is to lighten the potentially burdensome number of questions and grading that arise with such a large class. We hesitate to think about handling such a large number of students without group programming projects.

The results shown in the paper indicate that the course is successful and as part of working together the students are introduced to and certainly understand the need for good software engineering principles. Thus while the course does not provide a lot of formal instruction on these principles it helps motivates the importance of software engineering when students take the course later in our curriculum.

An outgrowth of our work has been sponsorship of an National Science Foundation-supported workshop on the application of peer learning to the introductory computer science curriculum [14, 15]. This project has brought together a diversity of computer science educators for an ongoing process on the application of peer learning to the introductory computer science curriculum.

References

[1] E. B. Koffman, D. Stemple, and C. E. Wardle, "Recommended curriculum for CS2: 1984 a report of the ACM curriculum task force for CS2," *SIGCSE Bulletin*, vol. 28, pp. 815–818, August 1985.

[2] ACM Curriculum Committee on Computer Science, "Curriculum 78: Recommendations for the undergraduate program in computer science," *Communications of the ACM*, vol. 22, pp. 147–166, March 1979.

[3] B. Hartfield, T. Winograd, and J. Bennett, "Learning HCI design: Mentoring project groups in a course on human-computer interaction," *SIGCSE Bulletin*, vol. 24, pp. 246–251, March 1992.

[4] H. Pournaghshband, "The students' problems in courses with team projects," *SIGCSE Bulletin*, vol. 21, pp. 37–41, February 1989.

[5] J. C. Prey, "Cooperative learning in an undergraduate computer science curriculum," in *ASEE/IEEE Frontiers in Education '95*, November 1995.

[6] H. M. Walker, "Collaborative learning: A case study for CS1 at Grinnell College and UT-Austin," *SIGCSE Bulletin*, vol. 29, pp. 209–213, March 1997.

[7] R. E. Sabin and E. P. Sabin, "Collaborative learning in an introductory computer science course," *SIGCSE Bulletin*, vol. 26, pp. 304–308, March 1994.

[8] E. Roberts, J. Lilly, and B. Rollins, "Using undergraduates as teaching assistants in introductory programming courses: An update on the Stanford experience," *SIGCSE Bulletin*, vol. 27, pp. 48–55, March 1995.

[9] C. E. Wills and D. Finkel, "Experience with peer learning in an introductory computer science course," *Computer Science Education*, vol. 5, no. 2, pp. 165–187, 1994.

[10] C. C. Bonwell and J. A. Eison, "Active learning (creating excitement in the classroom)," *ASHE-ERIC Higher Education Report 91-1*, 1991.

[11] S. I. Feldman, "Make—a program for maintaining computer programs," *Software—Practice and Experience*, vol. 9, pp. 255–265, March 1979.

[12] D. Finkel and C. E. Wills, "Computer supported peer learning in an introductory computer science course," in *ACM SIGCSE/SIGCUE Conference on Integrating Technology into Computer Science Education*, pp. 55–56, June 1996.

[13] C. E. Wills and D. Finkel, "Study of a group project model in computer science," in *IEEE/ASEE 1997 Frontiers in Education Conference*, November 1997.

[14] C. Wills, D. Cordes, D. Deremer, B. Klein, R. McCauley, and L. Null, "Application of peer learning to the introductory computer science curriculum," *SIGCSE Bulletin*, vol. 29, pp. 373–374, March 1997.

[15] C. E. Wills, "Application of peer learning to the introductory computer science curriculum." `http://www.cs.wpi.edu/~peercs`.