Engaging Large First Year Classes

Katrina Falkner
University of Adelaide, Adelaide, Australia
katrina.falkner@adelaide.edu.au

Abstract: *Engaging first year students is a difficult problem, as students must develop independent study skills while concurrently mastering their chosen topic. At the same time, they find themselves in an alien environment, removed from their peer group and anonymised by University structures. Collaborative learning is a strategy that involves the students themselves in the ownership and direction of their learning; each student is responsible for not only their own learning but of the learning of the group. In this paper we describe our approach to student engagement based on applying a range of collaborative learning techniques within an introductory programming course, addressing specifically the task of collaborative problem solving.*

Introduction

Engaging first year students is a difficult and well-recognised problem, as students must combine the development of independent study skills with mastery of their chosen topic. Although already a difficult prospect, this is further complicated by the transition to University education, requiring most students to enter an alien education system, consisting of large classes, a de-personalised administrative system and separation from their peer group. Studies indicate that students feel "part of an anonymous mass" (White, 2006) and indicate that one of the most common reasons for leaving the University education system is a feeling of isolation and loneliness (Burns, 1991).

The understanding of programming poses many challenges for first year students, as a large number enter ICT or Engineering programs with little or no prior programming experience. Novice programmers are required at an early stage to develop a diverse range of skills: problem analysis, problem solving, code development and testing, and then integrate those skills within the software development process. Studies show students continue to struggle with this task, even though it is common for Universities to dedicate a disproportionate amount of resources towards their learning (e.g., McCracken *et al*, 2001).

In this paper we describe the use of *Collaborative Learning* principles (Smith and MacGregor, 1992) in the teaching of an introductory programming course in order to increase student engagement. Collaborative learning defines a range of educational approaches that involve group work, with students being responsible for not only their own learning but also that of the group. In these approaches, the emphasis is constructivist, and based on the student's experiences in application of course content rather than in their observation of course content.

Collaborative Learning

Collaborative Learning encompasses a range of constructivist learning techniques where students work in groups, sharing and constructing their knowledge within a common aim. Collaborative learning is known to be effective in academic development, as students are more engaged with the subject matter, but also supports community and social development. It is seen as a particularly relevant technique in first year education, in that it addresses transition concerns by increasing deeper learning and a sense of belonging (McKinney and Denton, 2006).

Collaborative inquiry is supportive of learning for all parties involved, not simply through the sharing of knowledge but in the elaboration and discussion that ensues. Glasser (1990) draws a distinction between learning by observing and learning by doing and being involved:

> "Students learn 10% of what they read, 20% of what they hear, 30% of what they see, 50% of what they see and hear, 70% of what is discussed with others, 80% of what they experience personally, and 95% of what they teach to someone else"

Examples of collaborative learning include *cooperative learning* (Slavin. 1983), *discussion groups* (Christensen, Garvin and Sweet, 1991) and *peer teaching* (Jenkins and Jenkins, 1987). Cooperative learning is a heavily structured form of collaborative learning, where students participate in groups with a significant emphasis on interpersonal skills and assessment of management roles. This approach adopts cooperative incentive structures, where the success for the individual depends on the success of the group as a whole. Discussion groups can be either formal or informal and involve a discussion led by both the teacher and the students around a specific topic. The aim here is to open dialogue and explore an open-ended discussion, enabling each student to express their views and for students to debate and learn from each other's opinions. Peer Teaching, perhaps better named *cross-age teaching*, involves more advanced students acting as experts employed to instruct and mentor novice students. Although not strictly *peers*, the two students are generally close in age and share in the same experiences. There is no boundary of authority between the two, with no threat of assessment, leaving the novice able to openly express their concerns.

An essential aspect of a collaborative activity is the authenticity of the task that is to be attempted, and the processes that are applied. Mutual engagement requires that each member have a genuine motivation for being part of the collaboration, and that learning be associated with a task or problem that the group has to solve. We have selected problem solving as the task around which we construct our collaborations. Although the adoption of a single collaborative learning technique provides benefit, we propose that adopting a comprehensive programme of multiple collaborative learning techniques provides a more authentic collaborative learning experience as collaboration is then perceived as inherent to learning and the practice of the discipline.

Collaborative Learning of Programming

A comprehensive programme of collaborative learning techniques has been introduced in an introductory programming class in order to address the issues of student engagement and transition concerns. The introductory programming course of approximately 200-250 students contains a mixture of domestic and international students (approximately 50% of each), and is taken by students from a range of disciplines, including Computer Science, Software Engineering, Computer Systems Engineering, Electrical and Electronic Engineering, Telecommunications Engineering, Commerce and Arts. The majority of students who take this course are entering University straight from the secondary education system and are new to the process of adult, or self-directed, learning.

In an introductory programming class we wish students to learn different programming concepts, how to apply each concept and how to compose them to solve a larger problem. Problem solving skills are used both in the identification of the specific programming concepts to use in solving a problem, and in directing how programming concepts should be composed. Our main exploration of this process is through a series of one-hour lectures, held three times a week to all students in the class. In order to engage the students in the learning approach, the course starts with a discussion of the benefits of cooperative learning and related research (e.g. Glasser, 1990).

Lecture classes adopt a combination of observation using known-answer questions, and experiential collaborative learning using *Worked Examples* questions. These are designed to encompass the spectrum of the problem solving process:

- observing the application of programming concepts
- observing authentic problem solving
- cooperative problem solving

Traditional teaching approaches describe programming techniques via examples of code. Moving away from this methodology to the demonstration of programming enables students to observe what they will soon practice. The process is relatively simple: whenever we come across a code example in our lecture notes, we replace it with a live demonstration of developing said code. This enables us to demonstrate the composition of programming concepts, and the use of the laboratory environment. Our live demonstrations are performed using step-wise refinement; we discuss each step with the students and explain why we took the selected action. Students receive a documented description of the development process and resulting code in their lecture notes.

Demonstrating the application of programming concepts enables students to observe the construction of the building blocks of a software system. This is only part of learning the software development process; an understanding of the required problem solving skills is also essential.

We have integrated *Problem Solving* lectures at threshold points in our curriculum. At the stages where students must learn to compose programming concepts, and to solve more abstract problems, we insert a lecture dedicated to the demonstration of *live* problem solving. The process in these lectures is to start with a high level, and incomplete, problem specification, for example:

> *We have been contracted to develop a new mobile phone. We decide to build a software simulation in Java of how it is going to work.*

Problem solving lectures followed the following process: elaboration of the problem, identification of key elements of the solution, and conclude by identifying key classes and necessary algorithms.

The lecturer leads whole-class discussion at each stage, with individual students contributing questions and design ideas at requested points. An important aspect of these lectures is to explore the potential problem space, rather than focusing on one planned pathway. Multiple solutions, based on the inputs of the students, are presented. Two or more potential solutions are then taken from the design phase through to implementation, enabling discussion of the positives and negatives of each selected solution.

Our final problem solving component, and the most crucial in establishing authenticity and engagement, is cooperative problem solving. We introduce cooperative problem solving through weekly *Worked Examples* lectures where the students are presented with a small set of problems to solve. The problems are designed to utilise the programming concepts demonstrated in the previous lecture. The distinction that we make between these lectures and the Problem Solving lectures is that in these lectures the students lead the activity, and the lecturer acts as a scribe. Our initial approach to these lectures involved adopting a co-lecturing approach with two lecturers engaging the class in discussion to explore the problem space. Although the students found this an entertaining experience, we found that they were acting as observers and did not naturally take a leadership role.

Subsequently (in 2007), we adopted a small group cooperative approach. From the very first class, we organise students into small groups of 3 or 4 students. *Worked Examples* lectures then involve allocating 2 or 3 problems across the individual groups. Groups are then asked to discuss the problem to make sure that each member of the group understands what is being asked, to design a solution and to start to develop a software implementation of their design. The role of group scribe is rotated within the group throughout lectures, giving every student a chance - and the responsibility - to lead the process. In these classes, the lecturer moves between groups, advising them and answering their questions. The lecturer is then able to observe common problems, and to gather examples of different solutions. In the last 10 minutes of the lecture, the lecturer summarises common issues and presents one or two example designs. Multiple solutions to the problem are then posted to the electronic bulletin board following each lecture to enable the class to continue its discussion.

This stage enables the lecturer to make personal contact with each student (over the semester), and helps identify students who need additional assistance. We are able to identify individual students who required the individual consulting provided by the lecturers outside of class. Informal discussions can be held between individual groups and the lecturer; students reported that they felt more comfortable in asking questions and arguing their positions in their groups than they felt in larger classes with a tutor or lecturer. They also indicated that they obtained a clearer picture of where they stood relative to their peers, providing sufficient incentive for some students to seek further help outside of class in order to catch up.

Interestingly we found that students remained in their initial groups for the remainder of the semester, regardless of whether they were undertaking group or independent work. We feel this is of wider benefit in an introductory class; forming of bonds within the peer group at the commencement of University education addresses a common cause of transition distress (Krause *et al,* 2005).

Collaborative Support Systems

In addition to collaborative lectures, we introduced collaborative tutorial sessions, and support services for our introductory programming class. Weekly tutorials are held in a similar style to the Worked Examples lectures. Students are expected to prepare answers to a set of tutorial questions prior to the tutorial, which are then assessed by the tutor. Within the tutorial session, the students work in small groups to elaborate on their answers with each group working on a different problem. Each group then presents its solution to the rest of the tutorial class.

We introduced a training program for all casual academic staff involved in first year tutorials and practical sessions in 2006, providing an introduction to collaborative learning and small group teaching techniques. The structure of the training programme adopts collaborative learning techniques itself, in that participants learn and reflect on their teaching skills through the use of small group role-play sessions. All tutors and practical supervisors must attend this program, regardless of previous attendance. This provides further opportunity for reflection for the experienced participants, and the opportunity for cross-age teaching for the novices.

It can be difficult to provide appropriate support services for first year University students, as many are shy and lacking in confidence. Beder (1997) reports that first year students are simply not aware of support services on offer, and do not take advantage of them. We introduced the *Computer Science Learning Centre* as a dedicated space for cross-age teaching. The learning centre is a space positioned within the first year laboratory environment where novices may go for help on any aspect of studying Computer Science. The Learning Centre is staffed by senior undergraduate students who have undergone the same small group and cooperative learning training as our tutors. The Learning Centre staff act as informal tutors, providing help and advice on topics ranging from Java fundamentals to exam study techniques. The Computer Science Learning Centre is now the focal point for students within the School, with senior students often volunteering their time, beyond their scheduled hours.

Webb (1989) identifies six conditions that are needed for successful, and thus educational, cross-age teaching: (1) The tutor must provide relevant help which is (2) appropriately elaborated, (3) timely, and (4) understandable to the target student; (5) the tutor must provide an opportunity for the tutee to use the new information; and (6) the tutee must take advantage of that opportunity. As we are focusing on the very practical discipline of problem solving and programming, we have constructed the learning centre to reflect the laboratory environment of the student's normal practical sessions. The space is divided into two sections: one section containing computers for practical work, and the other a general study and discussion area. This model enables novices to move between teaching and application phases without leaving the space, and the care of the expert.

Analysis

Student perceptions on their ability and engagement with the course altered significantly subsequent to the adoption of cooperative learning activities within lecture sessions. Table 1 shows the results of student surveys showing broad agreement with a series of 7 point Likert style questions over this period, where broad agreement is the percentage of students who have responded positively (not neutrally) to the question.(Data is shown for 2005 (prior to the commencement of the collaborative learning techniques), for comparison, with 2008 data).

Student comments indicate that they appreciate the opportunity to work together within lectures, with the accompanying realisation of the application of programming theory, identifying the best aspects of the course as: “Worked examples because they are good, especially in small groups”, and “Examples which students do themselves together, then she explains. Inclusion of students into lectures.”.

Question	2005	2008
Q1: I feel part of a group committed to learning	39%	58%
Q2: This course stimulates my enthusiasm for further learning	68%	74%
Q3: I am motivated to learn in this course	59%	71%
Q4: This course helps me develop my thinking skills (eg. problem solving, analysis)	77%	88%
Q5: I understand the concepts presented in this course	61%	80%

Table 1: Student Survey results from 2005 and 2008.

An increase in engagement should ideally be reflected by an increase in ability to apply course concepts and demonstrate core knowledge. Figure 1 illustrates average student performance in practical examinations held throughout the course, where each student is assessed as an individual. The first practical examination (PE1) assessed fundamental computer awareness skills and the ability to use the practical laboratory, rather than the application of programming concepts or software development. Questions in this examination were relatively simple: requiring correction of simple syntax errors, demonstration of the ability to edit, compile and execute source code. Failure in this examination is used as an indicator of a lack of confidence in using the laboratory facilities, and of requiring assistance in understanding instructions, rather than of a lack of comprehension of course concepts. Live demonstration of the task of programming, including failure and experimentation provides crucial experience and confidence that students require at this early stage in their program. Figure 1 demonstrates a significant improvement once we introduced live demonstrations and authentic problem solving in 2006.

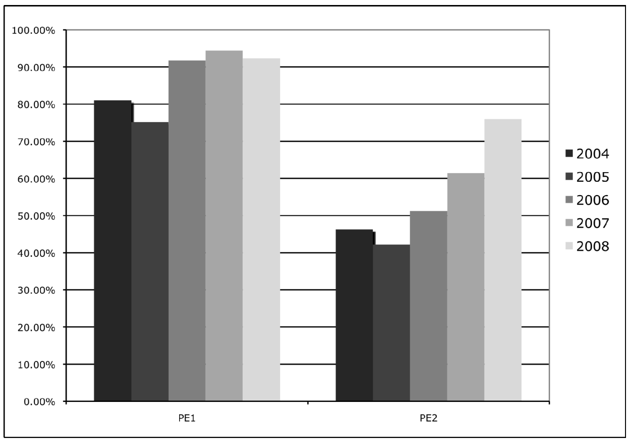

Figure 1: Practical examination performance from 2004-2008.

Figure 1 also demonstrates results for the second practical examination (PE2), which assesses problem solving skills and the application of programming concepts. As can been seen in the figure, the performance of students improved significantly once we introduced cooperative learning techniques (2007-2008), with the continued improvement attributed to the improved facilitation of the learning groups gained through our experience with this technique.

Out-of-class consulting is provided in each offering of the course, through in-office consulting hours for lecturers, and an on-line discussion forum. Since the introduction of the cooperative learning activities, students have engaged more with the course both within, and outside of, class. The number of students active on the discussion forums has increased from 44% to 93%, and the number of accesses averaged per student has increased from 4.9 to 191. Students have adopted the principles of cooperative and collaborative learning to create their own learning communities outside of the classroom, including establishing online problem solving forums where students set their own problems and rules for success, such as fastest execution time or earliest correct submission.

Interestingly, both cohorts (2007 and 2008) of students independently developed a variation on these forums in response to exposure to cooperative learning approaches.

Conclusions

In this paper we have described the application of collaborative learning techniques within an introductory computer science course. Our aim was to increase student engagement and student ability in problem solving through integrating a range of collaborative learning activities throughout the course structure, including lectures, tutorials and support services. These activities appeared to impact both the social aspect of the course, in that students were able to quickly develop a peer group, and also their ability to problem solve. The benefits of our integrated approach to collaborative learning can be seen both in terms of increased student confidence and perception of ability, and in increased student ability to complete problem solving activities.

References

Beder, S. (1997). Addressing the Issues of Social and Academic Integration for First Year Students: A Discussion Paper, *Ultibase Articles*.

Burns, R. (1991). Study and stress among first year students in an Australian University, *Higher Education Research & Development.* 16, 61-77.

Christensen, C.R., Garvin, D. and Sweet, A. (1991). *Education for Judgement: The Artistry of Discussion Leadership.* Harvard University Business School.

Ginat, D. (2008). Learning from Wrong and Creative Algorithm Design. In *Proceedings of SIGCSE 2008, Symposium on Computer Science Education*.

Glasser, W. (1990). *The Quality School: Managing Students without Coercion, 1st edition.* New York: Harper Collins, ISBN: 006055200X.

Jenkins, J.R. and Jenkins, L.M. (1987). Making Peer Tutoring Work, *Educational Leadership.* 44(6), 64-68.

Krause, K., Hartley, R., James, R. and McInnis, C. (2005). *The First Year Experience in Australian Universities: Findings from a decade of national studies.* Canberra: DEST.

McCracken, M., Almstrum, V., Diaz, D., Guzdial, M., Hagen, D., Kolikant, Y., Laxer, C., Thomas, L., Utting, I. and Wiusz, T. (2001). A Multi-National, Multi-Institutional Study of Assessment of Programming Skills of First-year CS Students, *SIGCSE Bulletin.* 33(4), 125-140.

McKinney, D. and Denton, L.F. (2006). Developing Collaborative Skills Early in the CS Curriculum in a Laboratory Environment. In *Proceedings of SIGCSE 2006, Symposium on Computer Science Education*.

Slavin, R.E. (1983). When Does Cooperative Learning Increase Student Achievement?, *Psychological Bulletin.* 94(3), 429-445.

Smith, B.L. and MacGregor, J.T. (1992). What is Collaborative Learning?. In A. Goodsell, M. Maher, V. Tinto, B.L. Smith and J.T. MacGregor (Eds.), *Collaborative Learning: A Sourcebook for Higher Education*. National Centre on Postsecondary Teaching, Learning and Assessment, Pennsylvania State University.

Webb, N.M. (1989). Peer Interaction and Learning in Small Groups. In N.M Webb (Ed), *Peer Interaction, Problem-Solving and Cognition: Multidisciplinary Perspectives*. New York: Pergamon Press. 21-29.

White, N.R. (2006). Tertiary Education in the Noughties: the student perspective, *Higher Education Research & Development.* 25(3), 231-246.