Computer Science Education
2000, Vol. 10, No. 2, pp. 109–128
0899-3408/00/1002–0109$15.00

Problem-Based Learning for Foundation Computer Science Courses

Michael Barg, Alan Fekete, Tony Greening, Owen Hollands, Judy Kay and Jeffrey H. Kingston
Basser Department of Computer Science, University of Sydney

Kate Crawford
SMITE Research Unit, Faculty of Education, University of Sydney

ABSTRACT

The foundation courses in computer science pose particular challenges for teacher and learner alike. This paper describes some of these challenges and how we have designed problem-based learning (PBL) courses to address them.

We discuss the particular problems we were keen to overcome: the purely technical focus of many courses; the problems of individual learning and the need to establish foundations in a range of areas which are important for computer science graduates. We then outline our course design, showing how we have created problem-based learning courses.

The paper reports our evaluation of the approach. This has two parts: assessment of a trial, with a three-year longitudinal follow-up of the students; reports of student learning improvement after we had become experienced in full implementation of PBL.

We conclude with a summary of our experience over three years of PBL teaching and discuss some of the pragmatic issues around introducing the radical change in teaching, maintaining staff support, and continuing refinement of our PBL teaching. We also discuss some of our approaches to the commonly acknowledged challenges of PBL teaching.

INTRODUCTION

Foundation courses in computer science pose particular challenges for the teacher: the courses develop basic skills and attitudes which are important for effective learning in later courses; they are often large courses with correspondingly large management and administrative loads; teaching staff often find them demanding and, for some staff, they are seen as onerous.

Correspondence: Judy Kay, Basser Dept. of Computer Science, Madsen FO9, University of Sydney, Australia 2006. E-mail: katecis@ozemail.com.au, or {mbarg, fekete, tony, owen, judy, jeff}@cs.usyd.edu.au.

Now consider the critical role of foundation courses from the learner's perspective. They give a large cohort of students their first real taste of the discipline. Negative experiences may discourage students from further study. This is a very serious problem if those negative experiences are not indicative of the discipline as a whole.

Consider an example. A large proportion of computer science graduates will find employment which involves broad problem-solving skills, rather than purely technically centered activity. If the foundation courses have a narrow technical focus, this may deter students with a preference for broad problem solving that is so much in demand among graduates. This has been identified as a particular problem for women (Kay et al., 1989).

Factors such as these weighed on us in 1995 as we began the redesign of our first-year courses. At that point, we, like many others, were still teaching Pascal. For several years, we had been acutely aware of Pascal's shortcomings but found the next choice of language very difficult. Even so, at this point, we felt that a move was urgent and we chose to move to an object-oriented programming (OOP) language. In fact, we moved to a clean, elegant object-oriented, introductory programming language, Blue (Kölling, 1999a, 1999b; Kölling & Rosenberg, 1996).

At the same time, we saw major problems with the overall teaching approach and structure of our foundation courses. Our foundation courses had a conventional format, with six contact hours per week, consisting of three lectures, a tutorial, and a two-hour workshop. We were concerned that this conventional course failed to develop skills needed in later courses.

The move to OOP provided additional impetus for re-examining the way that we taught. An important dimension of the OO paradigm comes from the writing of large programs which involve several programmers. If students do assignments alone, it is more difficult for them to appreciate this aspect. Certainly, one can provide large programs as a starting framework so that the individual work can have more of the flavor of typical OO projects, but this is less satisfactory than group development of a system.

THE CHOICE OF PROBLEM-BASED LEARNING (PBL)

As we refined our understanding of the problems with the conventional teaching framework, we identified the elements of a significantly improved structure. The teaching approach which seemed to offer most promise was

problem-based learning (PBL). Its literature indicates that it does develop just the range of learning we wanted. See, for example, Albanese and Mitchell (1993). PBL is most common in professional degrees, especially those in medical and paramedical areas. Seminal work in moving PBL to more technical disciplines has been in the area of chemical engineering (Woods, 1994; Woods & Sawchuk, 1993; Woods et al., 1975).

Before we describe what PBL is, we need to explain what it is not. We frequently find people who claim to use PBL but they appear to use this term in a different way from the large and well-established PBL community. For example, most computing courses involve setting "problems" which students are required to complete. We will refer to these as exercises, because they are small and well-defined. We used them extensively in our old conventional course: there were weekly exercises, each focused on particular detailed aspects of the course, usually one that had been center-stage in the recent lectures; there were larger assignments which integrated many aspects of the course but were still quite tightly defined, to the point where we could assess their correctness in our automatic grading system.

PBL involves much broader problems, which involve a larger set of problem solving skills. Critically, PBL places problem-solving and metacognitive skills at the heart of the curriculum. Class time is devoted to such generic problem-solving skills as defining a learning plan, brainstorming to get started on a problem, reflection, articulation of problems and solutions, self-assessment, practice in active listening, and other communication skills. These aspects are also assessed and contribute to the grade awarded.

Problem-based learning is learning by solving a large, real-world problem. Lectures are replaced by additional tutorial and laboratory time. In our case, staff are only present for one hour in the three-hour laboratory class. This is necessary to keep costs similar to those for a conventional course.

Critically, PBL courses actively teach generic problem-solving skills. We allocate teaching time to explicit instruction in various generic skills and we assess them. At the same time, we situate the learning within the context of computer science.

Examples of the problems we offer include simulating a road network, maintaining information about Olympic events and athletes, and answering arbitrarily complex database queries. The problems are open-ended, and groups are encouraged to research their subject and develop their own specifications and solutions. A different set of problems is set for the advanced class. All require research into techniques described in the computer science

literature. For example, here is one of the problems we offered in the first semester course:

> We are planning to open the Basser Software Mart. This problem requires assistance in planning the check-outs. Some people think that the best thing to do is have a single queue for customers and the person at the head of the queue goes to the first available check out. Other people think it is better to have an Express queue for two check-outs for customers who have less than 6 items and another queue for the remaining four check-outs for all customers.
>
> Your simulation will allow the planners to explore a range of scenarios. For example, customers arrive at different rates, with a burst of them a little after opening time, another burst during lunch time and yet another near closing time. Also, some days are busier than others. It would also be good to explore other possibilities, like cash-only queues.
>
> In your demonstration of the final project, you will show the simulation of two different ways of managing the queues and show how well each does in terms of things you consider important, like average wait time for service, maximum wait time or the like.

Such problems provide a driving force for developing metacognitive skills: students manage and monitor their own learning, and reflect on how to do this more effectively. The first attack on the problem involves students determining the following:

- problem statement—current understanding of overall goal(s)
- current subgoals
- how you will know you have succeeded
- what you already know
- steps to take, by when
- use of three-hour class time
- use of six-hour private study time

Finally, problem-based learning involves group learning. This serves as part of the development of skills in communication and cooperative work. It also means that students need to discuss their knowledge and approaches and to justify decisions: this externalization of understanding and knowledge is an aid to improved learning, especially in the case of metacognitive skills.

When we began designing our course, PBL was appealing. However, we were not able to take its pure form, where the students spend their whole degree program learning in this format. We needed to adapt it to our two foundation courses in an environment where most students spend only a quarter of their time in our PBL course, the other three-quarters being in various other courses.

In summary, PBL is characterized by:

- open-ended, authentic, substantial problems which drive the learning
- explicit teaching and assessment of generic and metacognitive skills
- collaborative learning in groups

OVERVIEW OF FOUNDATION COURSES

This section gives the flavor of the two courses by describing the way that activities are spread over the semester, and the issues associated with group work, assessment, and staff development. We describe the first semester in rather more detail so as to communicate more effectively our approach.

Semester 1: Introduction to Programming

The approach of the course is to view programming as building models which are implemented as classes, each instance of which corresponds to a real-world entity. The system will repeatedly analyze an event that corresponds to a real-world change, and it will evolve in response to this event.

The course objectives are formally set as follows. Essentially, they require that students be able to: define a "simple" class interface and implement it, making effective use of the Blue class library; use various code quality and testing strategies; reason about and explain the design of a systematic, economical and purposeful testing strategy; read and evaluate a class implementation in terms of modularity, code independence, class interfaces, class relationships, cohesion, coupling, overloading; utilize generic skills to plan learning, self-assess learning, use reference materials effectively, write an English report about the design of a "simple" class and its testing, give a well structured oral presentation about the design and testing of the system they have constructed; work cooperatively.

The expected outcomes are printed in the *Resource Book* (Kay, 1999) and class activities refer to them so that students are conscious of what we hope they will learn by the end of the semester.

There are three main periods in the first semester:

- weeks 1–4, the startup Problem 1
- weeks 5–11, main work on Problem 2
- weeks 12–13, reflective period, report writing, and demonstrations for Problem 2

During the first four weeks of the first semester, enrollment is too volatile to form stable groups for problem-solving. Instead, we present the first problem as a dry run for what is to come. The work in this period is assessed entirely individually. However, students work in groups, each group member doing different parts of the problem. A pass on Problem 1 is required before a student is allowed to join a Problem 2 group.

This period also involves several group activities in each week's tutorial. These involve a combination of generic problem-solving skills as well as technical skills. For example, an early activity involves practice in active listening with a partner who is doing problem-solving. We use generic materials taken from Woods (1994) but set them in the context of working out what code fragments do. We carefully choose code fragments: most students will need to work out the answers with the aid of the printed resources. The activity ends with reflection about problem-solving style and active listening skills, using Woods' questionnaires. Students are encouraged to change groups for each activity.

When students start Problem 2 in week 5, they know many in their class well enough to form groups which are reasonably compatible. We encourage students to form groups of people with diverse backgrounds, strengths and interests. At the same time, we offer a range of problems and the group has to agree on the problem it will select.

We have already given an example of a Problem 2 task with the Basser Software Mart. All the tasks require simulation of a complex system. All can be designed so that there is a small core of essential code in the main simulation driver classes. Once this is written and working, various other parts of the simulation can begin as very simple stub classes in a working system. These can be upgraded as group members implement more sophisticated versions. The tutors' task is to help guide students to a design which is safe for the group, because the essential core is implemented early and no individual student can prevent the group from producing a working system. (This task is not easy: we have been evolving strategies for assisting tutors in this role.)

The long duration of Problem 2 requires considerable student discipline if students are to be expected to work steadily, against weekly deadlines in other

subjects. We provide structure to the problem with several deadlines for the various stages of the problem solution.

In theory, a student in the PBL course could do a very similar sequence of tasks to those of a conventional course. Each week should have them doing an exercise in some aspect of the course. In parallel, they are working on the larger task. The difference between PBL students and their counterpart in our conventional course is that the former *decides* what tasks to do. They may get some assistance and guidance from the tutor in selecting a task from our resources or they may invent their own tasks.

The final "reflective" part of the semester is where students write reports on the problem solved by the group, present the demonstration, and write a reflective report on their learning, with their weekly plans as supporting evidence. Such reflective experiences are considered important to deep learning and are a standard part of the PBL approach.

Semester 2: Introduction to Computer Science

The second semester has two tasks, each running for half the semester. A common element in one of these is an outward focus on people ("users"), while the other is technocentric. As in semester 1, we offer choices in the problems for each task, taking care to offer tasks oriented to a range of interests, including business, life sciences, and engineering.

The outward-looking task has an information systems orientation. All choices involve inheritance as a fundamental aid for modeling the entities managed. This problem type makes it natural to incorporate ethical issues such as who might access or modify data. It also provides a perfect context for issues of scalability: so students need to assess the speed of the system as the amount of data grows.

Examples of the problem choices are: managing a biodiversity survey, managing information for an entertainment advisor, managing activities for Olympic participants, managing the data for a school timetable, and managing the product inventory for a computer vendor.

The second problem is intended to develop students' technical programming skills, especially recursion. Here we use parsing as the common theme in the set of problems offered. Each requires processing of some input whose format has been defined recursively. This makes a recursive implementation both natural and having a high pay-off. At the same time, the task introduces the use of classes which do not correspond directly to physical entities. It also provides a good context for use of the Composite object-oriented design pattern.

Choices of problems include developing: a spreadsheet including complex formulae defining cell values; a query interface (modeled on SQL) to tabular data; a pretty printer for a subset of the Blue language (Kölling & Rosenberg, 1996) used in the subject, an interpreter for a simple imperative language, or a compiler for that language.

Assessment

Assessment plays a critical role in any course. This and the role of criterion referenced assessment has been described by Biggs (1999). Essential elements of our approach are:

- criterion-based assessment, so we state carefully what we want student work to demonstrate
- equal marks for exam and practical work, so that students see that we value these equally
- within the practical work, equal marks for the individual and group work, so that students see we value both equally and so they are motivated to perform both individually and as a group member
- group assessment for tasks associated with group activity, such as group planning, management and coordination, and the group demonstrations
- individual assessment for code, according to the usual criteria for design, correctness, style, and documentation
- barriers in practical work where the individual must perform some tasks to an adequate standard before they may move on, and before they are entitled to the marks earned by their group
- minimum performance requirements for both examination and practical work, so that students cannot pass unless they achieve that minimum standard on both
- generic aspects are assessed on the examination as well as in the practical work, both because we want to assess learning in these areas and we want students to know we see them as important

Much of our work in this area is relevant to any course.

Staff Development

A move to PBL calls for a radical shift in the role of teaching staff. In a large first-year class, this means that introducing PBL requires a significant investment in staff development. We have addressed this in four main ways:

- literature about PBL
- staff development sessions
- scripts for teaching staff
- teaching mentors

The first is the easiest approach. However, it has limited value. First, it is difficult for people to make the time to read yet another set of papers. More importantly, as we found in the early stages of using PBL, the mental shift required is hard to achieve from merely reading papers. It is too easy to fall back on previous teaching and learning experiences.

We have found that intensive staff development sessions provide an excellent starting point. We run these for three days, with mornings devoted to classroom sessions and afternoons to practical activities. The morning sessions are generally attended by most of the staff, including experienced PBL teachers. These people can be spread through the groups used for most sessions. Less experienced staff are encouraged to do additional work in the afternoons.

Once the semester starts, we continue to support staff with detailed scripts. These map out the following:

- the preparation tutors should do for their classes
- an experienced teacher's assessment of the likely concerns students will have at this stage, especially where this relates to problems we know students will have accepting the strangeness of PBL
- tips for dealing with these problems
- how to run each activity, in terms of how long each aspect should take, what students should achieve in that time, what to do if students are not progressing as the schedule says they should, how difficult students typically find the activity, what resources support the activity
- explanations for our design of each of the activities, since we need the commitment of the teaching staff and we need to be sure they appreciate the purpose of learning activities

The first weeks of the first semester have very detailed scripts, up to eight pages long. As the weeks progress, the guidelines suggest more choices for the teacher to make and there is less detail. By the end of the first semester, the outline for a week's activity is usually two pages long, and in the second semester, formal scripts are not provided. Staff feedback indicates that this level of support is appreciated, especially by those teaching for the first time.

The final support comes from structuring the course into sections, with an experienced and committed person as section leader. This person has weekly meetings with the four or five tutors in their section. This gives new staff the opportunity to discuss problems that they and their students are having. The group can discuss ways to deal with these.

TRIAL IMPLEMENTATION

Since the shift from a conventional course to PBL is so radical, we first trialed PBL with a small group of students in 1996. Benefits of the trial were:

- careful evaluation of it was to inform the decision to move to PBL (or not)
- we needed an opportunity to learn how to run a PBL course, and a trial group was better for this than a large class
- when we were ready to move to full implementation of PBL, the trial provided solid answers to students and staff who were concerned about difficulties they encountered
- although we had not anticipated it, the trial results helped us when we met challenges in the full implementation and we, ourselves, sometimes questioned the wisdom of PBL

The trial involved an initial group of 42 students chosen randomly from a pool of volunteers; it was staffed by one of us and one regular CS1 tutor. It was structured similarly to the format we have described for the current course, although we have refined many details over the three years of full implementation.

Our trial implementation was complemented by an extensive evaluation undertaken by staff with professional expertise in the evaluation of higher education courses. Here we address three issues:

- assurance that the PBL students were not less competent at programming than main group students
- evidence that the additional generic skills that PBL is designed for were in fact being learnt
- data about students' attitudes to and perceptions of PBL, since these can have a marked effect on learning

For ethical reasons it was necessary to select the PBL stream from volunteers, so we cannot rule out self-selection bias altogether. However, we can say that the PBL group is typical of the main group in ability, as measured by the

Tertiary Entrance Rank (TER), a single number summarizing the performance of each student on the university entrance examinations. For the main group the average TER was 80.9 with standard deviation 9.2. For the PBL group the average TER was 81.2 with standard deviation 9.6, a difference that was not statistically significant at the 5% level.

Outcomes of the First Semester

Our analysis of the results of the first semester examination found no significant difference between the performance of the PBL group and either a group from the main CS1 class matched on the background variables of academic achievement and previous computing background, or the main class as a whole. When the results of individual examination questions were analyzed, one question was answered significantly better by main group students (not surprisingly, since that question related to a specific way of drawing data structures, as used in lectures, and this was not used in the PBL stream) and one question was answered significantly better by PBL students (for no clear reason). These results effectively answer concerns that eliminating lectures will reduce student learning.

We note that the examination was set by the lecturers of the main CS1 class. As one would expect, this meant that the particular approaches of those lecturers was reflected in the questions set. That the PBL students achieved the same levels of performance as the main CS1 class is quite a positive outcome for the PBL trial.

On the second issue, the learning of generic skills, we do not have quantitative comparisons between the main and PBL groups, but in many cases there is no basis for such comparisons. PBL students were required to design, plan, implement, test, manage, and report on a large group software project. No comparable demands were made of students in the main group. This is a crucial advantage of PBL: that, without losing any of the technical skills, it makes room in the course for activities that encourage generic skills. We would have liked to assess the generic problem-solving skills, as, for example, has been done in the case of an introductory engineering course (Reeves & Laffey, 1999). Our limited resources made this infeasible.

To address the third issue, student perceptions and attitudes, we conducted an open-ended questionnaire at the end of the first semester, in which we asked students to complete the statement “After one semester of computer science my attitude to this course is . . .” Answers were coded on two scales: how the students felt they were engaged in meaningful learning, and their emotional

response. The answers were analyzed by a graduate student from the Faculty of Education at our university, under the direction of the author from that faculty. The PBL students were compared against the matched group from the main CS1 class.

There is a marked difference between the responses from the PBL and main groups. A larger proportion of PBL students expressed a positive attitude to learning and a positive feeling about the course.

Longer-Term Follow-up

Funding for the detailed analysis of the trial covered the first semester, so the most detailed analyses are for that semester. We also compared the results for the second semester practical examination. This was a three-hour examination during which the students worked individually at a computer terminal on a previously unseen programming problem. Students who failed the examination had an opportunity to try again later.

The percentage of students passing the first sitting of the practical examination was 48.5 for the PBL group (with average marks of 55% and standard deviation of 40.5%), and 48.1 for the main group (with average marks of 58% and standard deviation of 38%). The differences are not statistically significant at the 5% level.

We analyzed the longer-term performance of the students in the PBL trial. Of the 42 students who began first year, 22 completed second-year studies and 16 studied third-year level subjects. These numbers represent a slightly higher retention to second-year level for the PBL group and then a slightly lower level for the PBL retention to third-year level. This might be consistent with a positive first-year experience, followed by a more negative perception of the second-year units, which were all taught in a conventional format. Perhaps more importantly, at second-year level, the PBL trial group were part of a large class dominated by those who had studied in the conventional format. Lecturers would have tended to be most responsive to the bulk of the class, perhaps being less aware of those from the PBL trial group. With such a small sample, care is needed interpreting statistical comparisons with the main class.

Analysis of examination results shows no significant difference between the trial group and the main class.

Affective Measures

In 1999, we undertook a limited study of the long-term affective aspects. Two groups of students were invited to complete an open-ended questionnaire.

One group was the 13 students who began first year in 1996 and were in the fourth-year honours class in 1999. This included 2 students from the PBL trial group. In addition, 5 had been in the first-semester advanced class which was in PBL format. The remaining 6 had been in the conventional course. The second group approached was those students from the 1996 first-year class who were still studying third-year level units in 1999. These students had taken more than the minimum time to reach third-year level. This may be due to a range of reasons. For example, some studied in combined degrees where it is normal to study third-year level subjects in the fourth year of study. Others had failed subjects or taken time away from their studies. All such students were mailed. Responses were collected from 6 third-year students: 4 from the conventional course, 1 from the PBL trial group, and 1who had been in the advanced first-semester course in PBL format. We cannot claim these two groups as representative of the class as a whole. However, they are diverse. Also, the students who study in the honours class are important as future researchers and top students.

The questionnaire asked open-ended questions like: “What do you remember about your first year of computer science?” “What do you think were the positive aspects?” “What do you think were the negative aspects?”

There were clear differences in the character of the answers from those who had studied in PBL format compared with those in the conventional course. For those in the conventional course, answers focused on the specific language and lectures. They recalled the positive aspects in the same terms, citing aspects like learning programming skills as a positive experience. Negative aspects were varied but involved machine resources and specific lecturers. However, one recalled that they “didn’t get to know many people”.

The students who had studied in the PBL format gave broader answers, including aspects like “friendship” and “group discussion” as well as working on “challenging problems”, “self-directed learning” and “group work”. Some commented on the self-paced and self-directed learning as a positive aspect.

Group work was cited with both positive and negative comments. Students liked the small, helpful, and friendly environment of PBL. On the other hand, one member of the trial group had extremely negative memories. Many PBL courses report a minority of students expressing this view. They also indicate the need for better scaffolding for students.

Negative aspects of PBL emphasized the lack of guidelines and feedback, lack of structure, and the uncertainty as to what exactly was expected. Other comments included having problems with the structured exam.

A largely consistent picture emerges for the students who had studied in the semester 1 PBL format advanced class. They liked the intellectual challenges and open-endedness of PBL. They identified group work as useful and pleasurable but also as a source of frustration.

EXPERIENCES FROM FULL IMPLEMENTATION

In three years of full implementation, we have refined the courses considerably. One would have expected this with any new course, even ones which only involved a move to a new programming language. However, with the radical shifts involved with the move to PBL, there has been more learning on our part.

For example, the assessment criteria are particularly important in our PBL courses. A conventional course can get by with vague assessment criteria. In PBL, with the very general problem statements as a starting-point, it is all the more critical to give students a very clear understanding of how they will be assessed. One of the well-acknowledged problems that students report in PBL is that they are unsure how much they need to know, how far to explore their problems, and when they can reasonably stop learning. We have addressed this difficulty with a carefully crafted set of assessment requirements. We continue to improve aspects like this as we gain experience.

At this stage, we have evaluated several aspects of the courses. We now report on these.

Learning Outcomes

The following sets of graphs compare the results of second-semester examinations in the last non-PBL year (1996) with corresponding results in the second full PBL year (1998). There has been a substantial improvement in basic programming competence (Fig. 1).

The two questions are list traversal problems; the questions are different but they are of similar difficulty. We believe that this improvement is partly due to our new software, the Blue programming environment (Kölling, 1999a, 1999b). It is also partly due to the new curriculum, which returned programming to the center of the second semester unit of study.

Improvements have also occurred in questions on topics often thought to be beyond the grasp of the average student, such as time complexity analysis (Fig. 2).

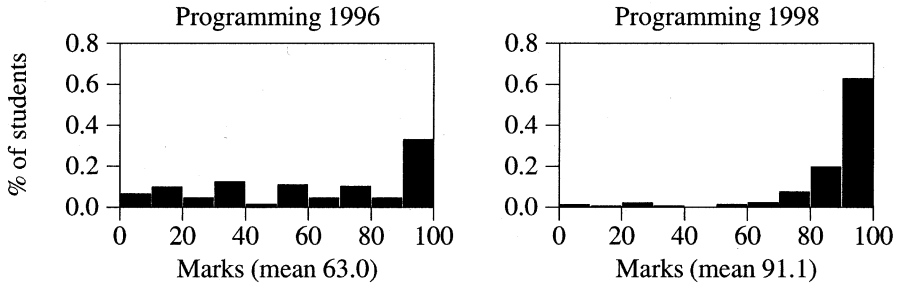

Fig. 1. Second-semester examinations results in the last non-PBL year compared with corresponding results in the second full PBL year.

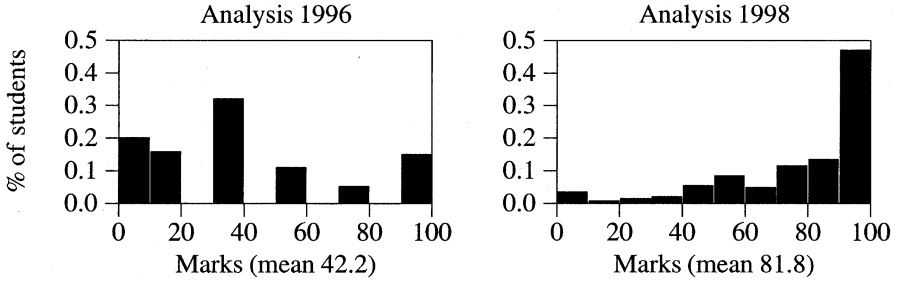

Fig. 2. Comparison of examination results on time complexity analysis.

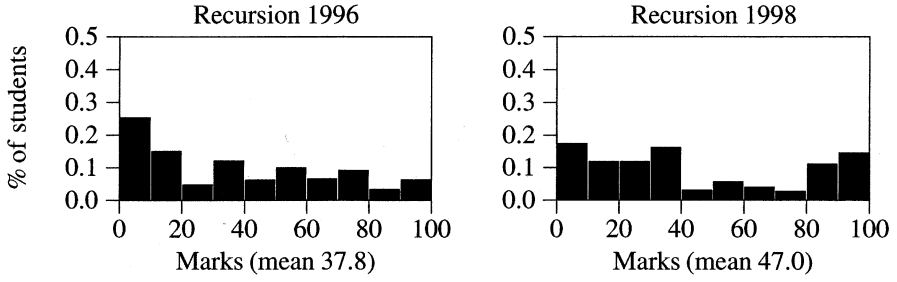

Fig. 3. Comparison of examination results on recursion.

These questions again are of similar difficulty. Another advanced topic, and a key skill for students progressing to second year, is recursion (Fig. 3).

Although we clearly have more work to do here, the 1998 recursion question asks the students to write a recursive descent parser, and thus is considerably more difficult than the tree traversal question from 1996. These improvements have occurred because these topics are now integrated into the

second-semester problems, whereas in the old unit they were buried in lectures which were poorly attended and perceived as irrelevant by many students.

Affective Aspects

Each year, we survey students in the middle and at the end of the first-semester course and at the end of the second semester course. Students indicate satisfaction with most aspects of the course. The seminars consistently rate less well than tutorials and workshops. We have revised the content and style of seminars over the years and student satisfaction has steadily improved.

RELATED WORK

Our adaptation of a relatively pure form of PBL to teaching foundation computer science courses seems to be novel. However, there are many elements of other computer science courses which share some features of our approach.

For example, about ten years ago, there was an innovative problem-based learning course in the Bachelor of Informatics degree at Griffith University (Little & Margetson, 1989). It had a very different emphasis, with the social context of computing being equal to the technical core. This course eventually ended, due to various difficulties, including the departure of some key staff members and antipathy to PBL by others. We have tried to learn from this effort. Our trial was important for building staff support. We have also devoted considerable effort to staff development, both in terms of improving the teaching and improving staff acceptance.

We have also been able to learn from the applications of PBL to engineering programs such as Woods (1994) and more recent cases like Reeves and Laffey (1999). Although much of that work is not directly applicable, it is closer to our situation than the larger-scale use of PBL in the medical and paramedical areas.

There has also been a computer science course taught successfully in an aeronautical engineering university (Hirmanpour et al., 1995). In that case, the relative consistency in the student population makes the course design properly directed at the needs of aeronautical engineering. This appears to have helped students appreciate the usefulness of this material.

More senior-level courses commonly have various degrees of PBL flavor. In particular, most computer science majors do a senior-level course in what is often called software engineering or a capstone project. We emphasize that

a basic principle underlying such courses is that the students have learned the basic technical skills needed for the project task: the goal of the course is to give them the opportunity to gain practice in integrating those skills learned in various courses. This is quite different from PBL, where the problem is the driving force for new integrated learning, not just the integration of existing knowledge. In practice, of course, we find that such courses do involve considerable learning of new skills required for the task, as well as building on existing knowledge. Such courses typically require and develop generic skills in communication and problem-solving, though the explicit support of that learning may not be provided. In practice, if not in philosophy, these senior courses have a strong PBL flavor.

Artificial intelligence has also been taught using problem-based learning (Cavedon et al., 1997). Several other recent initiatives in computer science education (Bareiss, 1996; Carver et al., 1996; Dietrich & Urban, 1996; Kirsch, 1996; Towhidnejad & Aman, 1996) have elements of problem-based learning. To our knowledge, however, we are the first group to create an entirely problem-based first-year computer science course.

CONCLUSIONS

We adopted problem-based learning, not for its own sake, but because we saw that it would foster the following goals.

An integrated curriculum. The educational literature warns against compartmentalized units of study that produce students who cannot integrate the different parts of their knowledge. Although a fully integrated degree was beyond our scope, the earlier conventional foundation courses had compartments that bore out the literature's predictions. The new courses are fully integrated, since students bring all their knowledge to bear on solving a few large problems.

Competence in computer programming, using a modern object-oriented programming language and environment. This is our main technical goal and the one students are most motivated to achieve. We have made this goal central, by structuring the units as a sequence of large programming problems, and requiring our students to use the Blue programming language and environment.

Ability to enhance program quality by using formal methods. This goal suffered from compartmentalization in the old units. We have integrated it with

programming by adopting a programming language with features to support it, using them routinely ourselves, and requiring our students to do so.

Ability to analyze the running time of programs and to produce programs with low running times. This was another victim of compartmentalization in the old units. One of our problems requires handling large quantities of data, and so students must master these techniques if their programs are to be efficient. We support their learning with tutorial exercises, a text book chapter, and a seminar.

Ability to use recursion. Recursion is an important conceptual tool and programming technique, and traditionally a difficult topic for novice programmers. One of our problems is inherently recursive, so that students must master recursion in order to solve it. We support their learning with several textbook chapters, seminars, and tutorials.

Independence and initiative. These are important generic goals because the old units were widely criticized for stifling them. The problems we offer are loosely specified, leaving plenty of scope for student initiative. Students discover they can learn by themselves, using a range of resources. They are aided in learning to do this by the PBL planning structures and tutorials which develop metacognitive skills.

Critical thinking and problem-solving. These are crucial to effective programming, especially at the higher levels of analysis and design. We ensure that students encounter these higher levels by having large projects requiring careful analysis and design. One of our tutors' most important roles is to guide students in learning these skills as they work on the problems. We also provide tutorials to help students develop generic problem-solving skills.

The ability to work in a group. This is important for employers; successful groups also increase students' confidence and initiative. All our problems after the first four weeks are group problems. We support groups by identifying specific roles for group members, providing class time and guidelines on group management, monitoring group planning and progress, and assigning marks for group management and reflection on group processes.

Communication skills. This is another goal highly valued by employers. Students working in a group naturally learn to communicate with one another. At the end of each problem the students give a demonstration, during which each student must speak, and present a written report. We also require appropriate English commentary within the students' programs.

Planning. Real-world programming projects are very large, and most failures are due to poor management rather than technical problems. To expose

our students to these issues, we set large problems (7–10 weeks' work by a group of four or more students) and assess group and individual planning as well as product.

PBL fosters generic skills such as group work—also necessary for a full appreciation of object-oriented programming—planning, problem-solving, independent learning, research skills, writing, and oral presentation. These are university goals and also highly valued by employers in the computing industry. PBL allows students to achieve far more in their programming than was possible within the small, individual assignments of the old units; and it virtually eliminates plagiarism, a long-standing problem in the conventional course.

ACKNOWLEDGEMENTS

The introduction of problem-based learning for our large first-year classes required many important contributions. There have been university and faculty grants: Teaching Quality Grant 1995, Student Progression Assistance Scheme Award 1997–8, and Information Technology Committee Grant 1998–9. Many departmental staff have been involved; particular support was provided by Professor John Rosenberg and Dr. Michael Kölling.

REFERENCES

Albanese, M. A., & Mitchell, S. A. (1993). Problem based learning: A review of literature on its outcomes and implementation issues. *Academic Medicin*e, *68*, 52–81.

Bareiss, C. C. (1996). A semester project for CS1. In *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, pp. 310–314.

Biggs, J. (1999). What the student does: Teaching for enhanced learning. *Higher Education Research and Developmen*t, *18*, 57–75.

Carver, C. A., Howard, R. A., & Lane, W. D. (1996). A methodology for active, student controlled learning: Motivating our weakest students. In *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, pp. 195–199.

Cavedon, L., Harland, J., & Padgham, L. (1997). Problem based learning with technological support in an AI subject: Description and evaluation. In *Proceedings of the Second Australian Conference on Computer Science Education*, pp. 191–200.

Dietrich, S. W., & Urban, S. D. (1996). Database theory in practice: Learning from cooperative group projects. In *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, pp. 112–116.

Hirmanpour, I., Hilburn, T., & Kornecki, A. (1995). A domain centered curriculum: An alternative approach to computing curriculum. In *Proceedings of the 26th SIGCSE Technical Symposium on Computer Science Education*, pp. 126–130.

Kay, J. (1999). *1001/1901 Resource Book*. Sydney: Basser Department of Computer Science.

Kay, J., Lublin, J., Poiner, G., & Prosser, M. (1989). Not even well begun: Women in computing courses. *Higher Education*, *18*, 511–527.

Kirsch, R. P. (1996). Teaching OLE automation: A problem based learning approach. In *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, pp. 68–72.

Kölling, M. (1999a). Teaching object orientation with the Blue environment. *Journal of Object-Oriented Programming*, *12* (2), May, 14–23.

Kölling, M. (1999b). The Blue language. *Journal of Object-Oriented Programming*, *12* (1), March/April, 10–17.

Kölling, M., & Rosenberg, J. (1996). Blue: A language for teaching object-oriented programming. In *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, pp. 190–194.

Little, S. E., & Margetson, D. (1989). A project-based approach to information systems design for undergraduates. *Australian Computer Journal*, *21*, 130–138.

Reeves, T. C., & Laffey, J. M. (1999). Design, assessment, and evaluation of a problem-based learning environment in undergraduate engineering. *Higher Education Research and Development*, *18*, 233–246.

Towhidnejad, M., & Aman, J. R. (1996). Software engineering emphasis in advanced classes. In *Proceedings 27th SIGCSE Technical Symposium on Computer Science Education*, pp. 210–213.

Woods, D. R., Wright, J. D., Hoffman, T. W., Swartman, R. K., & Doig, I. D. (1975). Teaching problem solving skills. *Engineering Education*, *66*, 238–243.

Woods, D. R., & Sawchuk, R. J. (1993). Fundamentals of chemical engineering education. *Chemical Engineering Education*, 80–85.

Woods, D. R. (1994). *Problem-based learning: How to gain the most from PBL*. Hamilton, Ontario: McMaster University Bookshop.