Citation:
Hamilton, M, Harland, J and Padgham, L 2003, 'Experiences in teaching computing theory via aspects of problem-based learning', in T. Greening and R. Lister (ed.) Proceedings of the Fifth Australasian Computing Education Conference (ACE2003), Sydney, Australia, 4-7 February 2003, pp. 1-5.

Experiences in Teaching Computing Theory via Aspects of Problem-based Learning

Margaret Hamilton **James Harland** **Lin Padgham**

School of Computer Science and Information Technology
RMIT University,
GPO Box 2476V, Melbourne, 3001, Australia
Email: {mh,jah,linpa}@cs.rmit.edu.au

Abstract

Computing Theory is a compulsory subject that many computer science students find difficult, and for which there is a wide range of abilities and backgrounds amongst students. In this paper we describe the evolution of this subject at our university over the past decade, which has resulted in a much more *student-centred* approach to learning. This has not only enhanced the learning experience of the students but also has simplified the development of teaching modes and resources involving on-line replacements for lectures.

1 Introduction

Most undergraduate courses in computer science include some compulsory material on the theory of computation. AT RMIT, the compulsory subject known as Computing Theory forms part of the second year of a three-year computer science degree. From the students' perspective, this subject is widely considered to be conceptually difficult, and its relevance can be hard to ascertain, especially when compared with foundational programming subjects (which the students have typically just completed and whose relevance to their studies can be clearly seen). For these reasons, in addition to its istorical and theoretical content, many students dismiss the subject in advance as boring without much thought, putting the teaching staff in the position of continually having to dispel this perception. Also, the wide range of abilities and cultural backgrounds amongst the students means that it can be difficult to know the appropriate level at which to "pitch" the material. This problem is particularly acute for the level of mathematical ability that the students are required to display.

The traditional approach to subjects like this has been to introduce conceptual material in lectures, followed by some exercises and examples in tutorials, and possibly by practical work in a laboratory session. In this sense the tutorials and laboratory sessions are supplementary to the lectures, which form the main learning engine for the subject. This was the way in which this subject was taught at its inception at RMIT in 1993, and for which the assessment was an exam worth 70% of the overall mark, and three assignments, each worth 10% of the final mark.

Since that time, our teaching methods have evolved significantly, in order to make the learning process more *student-centred*. In 2003, students will allocate themselves to *learning groups* of around 6 people, work on problems in these groups in the spirit of *problem-based learning (PBL)(1 , 2 , 6)*, and undergo a wide range of assessment methods (an exam, two assignments, an essay, a mid-semester test and self and peer assessment). In addition, students will have one two-hour tutorial per week (in place of the more normal one hour per week) as well as the standard weekly allocation of two hours of lectures and a one-hour laboratory session.

Much of this approach has developed from some earlier work on the application of PBL techniques to an artificial intelligence subject (3). PBL was originally developed to improve the quality of medical education by shifting from a strictly subject-based academic curriculum to one based on solving real life problems which may cross traditional discipline boundaries. In this paper, we describe the evolution of this computing theory subject over the past nine years and discuss our conclusions based on this teaching and learning experience. We also discuss how, unexpectedly, this method of teaching can be useful for on-line delivery of this material.

This paper is organised as follows: in §2 we discuss background issues and in §3 the main aspects of our current approach. In §4 we discuss the approach to tutorials and the role of the tutor, and in §5 our approach to assessment. In §6 we discuss the application of this approach to on-line learning and in §7 we present our conclusions.

2 Background

2.1 Material

The material taught in this subject is fairly standard in computer science courses, and there are a large number of textbooks which cover the relevant topics. These include finite state automata, grammars, parsing, regular expressions, Turing machines, computability, complexity and cryptography. A certain amount of both mathematical and programming ability is necessary, as evidenced by the pre-requisites which are a first-year discrete mathematics subject and two semesters of Java programming.

As it is not a mathematics subject, it is probably unreasonable to expect the students to be able to produce mathematical proofs (such as proving certain problems undecidable, showing the equivalence of non-deterministic and deterministic finite state automata, or finding reductions from one NP-complete problem to another); however, some mathematical basis is necessary, such as being able to follow proofs by induction (e.g. showing the equivalence of push-down automata and context-free grammars) or proofs by contradiction (e.g. the undecidability of the halting problem).

On completion of the subject, students will have demonstrated both conceptual understanding and the acquisition of new skills. The conceptual understanding includes terminology such as tractable, intractable

and undecidable problems, the relationships between elements of the Chomsky hierarchy (e.g. the equivalence between pushdown automata and context-free grammars) and the difference between approximation and probabilistic algorithms. The new skills include the ability to convert a nondeterministic finite state automaton into an equivalent deterministic one, to predict the behaviour of a given context-free grammar and to synthesize a Turing machine to solve a given problem.

The material is thus mathematically based, but is not mathematics per se. In particular, this means that students are expected to be able to follow proofs, but not necessarily to be able to synthesize them. [1]

2.2 Context

Since this subject is taught as a compulsory second-year subject in a three-year computer science degree, in a typical year there are around 350 students spread unevenly across two campuses. This generally means that lectures can be quite large, with around 300 students on one campus. This also means that there is a very diverse student population. The vast majority of students are undergraduates studying for a computer science degree, with a significant percentage of these students being international students and/or coming from a non-English speaking background. This subject is one of a number of compulsory subjects in our computer science degrees, and hence its flavour is very much a broad introduction to various important areas, rather than a specialised elective chosen by students inherently interested in the topics. As a result it is easy for the material to appear to be "bitsy" rather than integrated, especially as this integration is often only apparent in hindsight or when applying known techniques to an assignment problem.

3 General Approach

This subject was taught in the traditional lecture & tutorial mode for five years. The top-down approach of introducing material in lectures, reinforcing it in tutorials and applying it in assignment problems often leads to a lack of interest in the subject, partially because of the conceptually difficult nature of the material, and partially because the applications of it are not clear in advance. The large lecture sizes make it more difficult to interact with students in class. In addition, it is often tempting for tutorials to lapse into a mini-lecture mentality, with students asking few questions and hoping to be able to copy down solutions rather than work through the given problems.

As discussed in (5) (which describes the use of problem-based learning for introductory programming at the University of Sydney), a teaching method based on problem-based learning can be used to overcome some of these difficulties, and so in 1998 we revamped the way that tutorials were organised, taking our inspiration from problem-based learning. "Problem-based learning is an instructional method that challenges students to 'learn to learn', working cooperatively in groups to seek solutions to real world problems."(7) Under this model, students organise themselves into permanent working groups of (approximately) six, and work cooperatively on the problems set for each week. This encourages the students to engage the material for themselves, rather than to wait for someone else to talk them through it. Not only does this appear to make the process more enjoyable for students, it is also a more satisfying method of interaction for the tutors. It is emphasised to students that whilst we still maintain lectures (and laboratory classes) in the more usual way, the tutorials are now the main learning engine. This is further emphasised by making the tutorials two hours long; in many cases, one hour is simply insufficient to get students working in groups, engaging with the material and making sufficient progress each week.

A further aspect that underlines the importance of tutorials is that a portion of the overall mark (20%) is given for participation in the learning group. This is assessed on an individual basis by means of self and peer assessment. At the end of the semester, during the final tutorial, each student is asked to bring a written asessment of each of the members of the learning group, including themselves. These individual assessments are then discussed in the learning group until a consensus is reached. It is stressed to the students that this is about process and not achievement: making a significant contribution to the learning of the group is not the same as being able to easily answer all the set questions in a given week.

Our approach has been to modify the delivery of the curriculum over a period of years for the whole class. This is unlike the Sydney University trial mentioned above, in which a group of volunteers was selected to compare approaches. Our change has been gradual and the resulting improvements have been qualitative in nature.

4 Tutorials

For many students, it can be difficult to work in groups, and so the first week of tutorials usually involves some "ice-breaking" activities, such as finding the student whose birthday is nearest your own, or playing games which require memorization of the names of the other people in the tutorial. Engineering students, or at least those pursuing double degrees in computer sicence and engineering university, tend to be more practiced at group work, and so there is generally less need for such activities amongst these students. However, groups of friends or associates from outside the subject are not necessarily encouraged to stay in the one group, as each student is expected to contribute individually to the groupwork during the tutorial.

One key aspect of this kind of tutorial is that the role of the tutor is changed. "The principal idea behind PBL is that the starting point for learning should be a problem, a query or a puzzle that the learner wishes to solve." (8) Moreover it is important that students be guided towards solutions, and not driven towards them. In the words of a former student in Computing Theory "you have to try the wrong ways first in order to know why the right way is right". This means that tutors must be prepared to let each group make a reasonable attempt at the set problems before intervening to offer advice or encouragement, and to deal with any group management issues. It can also mean that the same explanations are made to several groups at different times; however, this is seen as a strength rather than a drawback, as each group will make progress at its own rate. A key rule that is enforced is that tutors are not to use whiteboards for explanations, even when this may seem useful to say something once rather than many times, as doing so tends to derail the group interactions and turn the tutorial into a mini-lecture. As a result, the tutors need to know what the goal is for each week, and to be prepared to give increasingly explicit guidance as the time during the tutorial wears on.

Our experience has indicated that this kind of tutorial has been very successful. It is not uncommon to students to make comments (unsolicited) such as "I

[1] The sole exception to this is the Pumping Lemma for regular languages, for which all proofs follow a very specific form, and only certain pieces of data vary from proof to proof.

wasn't going to take tutorials seriously until I realised that there was assessment involved. That made me turn up regularly, and I both learned a lot and enjoyed them". Students are generally more forthcoming about their lack of knowledge to others students than they are to a lecturer. It is also generally more enjoyable for the tutors as well. In some cases, tutors find this approach less daunting than having to be the "sage on the stage", as the kind of guidance needed is not so much "Here is the answer to the question you asked" as "Have you tried this approach to the problem?".

As discussed in (4), there is a tension between the discovery processes inherent in problem-based learning and the educational requirements of the subject, in that there is an expected starting point and finishing point based on the pre- and post-requisite subjects. Hence it is important to have a rough schedule for student progress, particularly so that the tutors have a reasonably good idea about what level of advice is appropriate at each point in the process.

5 Teaching Materials

Below are the main specifications of the two assignments used in this subject in 2002. In both cases the students had to submit a written report, and for the second assignment, their flex and bison code as well.

Assignment 1: You have recently been employed as a computer consultant to assist Hermes Logistics, a courier company. It is necessary for Hermes Logistics to ensure that each courier is given an optimal route in which to visit each of the destinations. A set of jobs for the routing to be optimized on can contain anywhere from 3 to 250 delivery destinations.

Hermes Logistics recently employed a team of highly paid consultants to solve this problem for them. The consultants produced a number of different programs (each using a different algorithm) for this, as well as accompanying documentation for each. However, they did not specify which documentation went with which program, and they took the source code with them. Hence Hermes Logistics does not know which program is the best solution for their problem.

Your task is twofold:

- Identify each program. The executable code and documentation can be found in `[Irrelevant Path Name]/Ass1`. You are to perform an analysis of the behaviour of each program on various input data in order to match it up with the relevant description.
- Recommend a solution to Hermes Logistics. For this you will need to give evidence as to why your chosen solution is better than all the others.

At the heart of the second task is the trade-off between execution time and accuracy. Some programs, it is claimed, will always find the fastest route and, their proponents argue, should therefore be the program used. Proponents of other programs claim that this approach is too slow and that even if the company buys a large amount of new equipment it will not cope with the number of jobs to be managed. Hence less precise solutions are used, which, it is claimed, will better meet the company's needs. Critics of this approach claim this too often produces non-optimal routing and that this will cost the company large amounts of money.

Management is willing to buy as much new equipment as necessary. They currently have capital which can be spent on equipment and reason that they are better off to buy any machines they need now, in order to save money eventually on courier wages. They are also willing to consider solutions such as waiting for a day to assign couriers their routes, in order to allow up to 24 hours of computation time to find a solution, if necessary.

Hence you must write a report which not only gives your identification of the programs and your recommendation but also carefully explains your reasoning. Your report should enable Hermes Logistics to fully understand the situation, with particular reference to the current state-of-the-art in this area.

Assignment 2: You have just joined a company named Smart Parsers who specialise in the development of compilers for programming languages. Your task is to develop a parser for the new programming language *Cappuccino*, which will check that a given program is correctly formatted (but does not need to do anything more complex, such as attempt to correct invalid syntax, or to generate parse trees or code).

This parser is to be built using the tools *flex* and *bison*, which are the standard parser generation tools at Smart Parsers. For the purposes of this assignment, you will need to determine how to specify the language that the parser should recognise without writing large slabs of code.

Documentation on Flex and Bison can be found in the notes package or at `http://dinosaur.compilertools.net`.

Example files which your program should be able to parse will be available from `[Irrelevant Path Name]/Ass2`.

Below is the specification of the syntax of Cappuccino.

- All *Cappuccino* programs must commence with `order(id)` where id is an identifier, and conclude with `drink`, which may be optionally followed by `sugar`.
- Apart from the above constructs a *Cappuccino* program consists of a *declaration_section* followed by a *statements_section*, and nothing else.
- A *declaration_section* commences with `ingredients`, is optionally terminated by `stneidergni`, and contains number of *variable_declarations.*
- A *variable_declaration* is either of the keywords `natural` or `boolean` followed by an *identifier list* followed by ';'.
- An *identifier_list* is either a single *identifier* or a number of *identifiers* separated by ','.
- An *identifier* is a sequence of alphanumeric characters, including lower- and upper-case, commencing with a lower case letter.
- A *statements_section* commences with `recipes`, concludes with `sepicer` and contains at least one *basic_statement.*
- A *basic_statement* is one of:
  - `brew`*(idnum)*
  - `steam`*(idnum)*
  - `pour`
  - `froth`

  - `smile`
  - `whenever` *boolean_expression* `proceed` *statements_section* `otherwise` *statements_section*
  - `repeat` *statements_section* `until` *boolean_expression*
  - `cup` *identifier statements_section* `puc`
  - `mug` *[identifier] statements_section* `gum`

  where *idnum* is either an identifier or a natural number.
- A *boolean_expression* is either
  - an *identifier*
  - of the form *identifier comparison_operator numerical_expression*
  - of the form *boolean_expression logical_operator boolean_expression*
- A *numerical_expression* is either
  - an *identifier*
  - an *natural number*
  - of the form *numerical_expression numerical_operator numerical_expression*
- A *comparison_operator* is one of =, > and <.
- A *numerical_operator* is one of +,-, *, /.
- A *logical_operator* is one of `&`, `|` and `!`.

For example, here are two legal *Cappuccino* programs.

```
order(fred)
ingredients
   natural x;
stneidergni
recipes
     steam(20)
     froth
     pour
     smile
sepicer
drink
```

```
order(jane)
ingredients
   natural x,p20,z,qQ1;
   boolean really,how,sad;
recipes
     brew(10)
     steam(30)
     froth
     froth
     cup{mary}
         recipes
            mug[john]
              recipes
              smile
              sepicer
            gum
         sepicer
     puc
     pour
     smile
sepicer
drink
sugar
```

Another problem that has been particularly useful is the following one.

"River Bank, a worldwide financial organisation, installs an RSA-based Web transaction server, and employs you as a software consultant, given your expertise in cryptography. One day, an officer of the bank notices a Web page giving a linear algorithm for factorization, and passes the information on to you, asking you to prepare a report on the consequences of this development for River Bank. What should the report say? What steps would you follow to work out what is to be said?"

6 Assessment

As mentioned above, self and peer assessment of the individual student's contribution to their learning group forms part of the overall mark for the subject. The assessment consists of:

- Exam (35-40%)
- Mid-semester test (0-5%)
- Two assignments (10% each)
- Essay (20%)
- Self and Peer assessment (20%)

The essay, which can be either a summary of the material studied during the semester or a detailed report of a specialised topic, is something that often causes some angst amongst the students, as they generally do not write many essays during their time as computer science students. However, it is a very useful way for students to demonstrate their holistic understanding of the material as well as to practise their technical writing skills. It is common to hear comments such as "I really didn't like the idea of writing an essay and I found it to be hard work, but I learnt a lot from doing so and I was glad, in the end, that I wrote it". This is particularly so for the summary essay. The specialist topic, for which the vast majority of students choose cryptography, is also a means of allowing students to do some fairly in-depth research on a topic of their choice.

One issue that can arise here is the distinction between exercises and assignments. Originally, we had the students work on assignment material during tutorials, in order to allow for greater and more timely feedback on assessment material and to allow for larger and more realistic assignments to be attempted. However, there was a stark drop in tutorial attendance when the final assignment had been submitted (which was a significant time before the end of the semester, in order to allow marking to take place and for the students to work on their essays). To solve this problem, the next year we made sure that there was some assignment work set for each tutorial. However, this created pragmatic difficulties with marking a large body of work at the end of the semester. Hence we have since reverted to the more traditional arrangement of doing exercises (which are not assessed) during tutorials and assignments (which are assessed) outside these times.

The self and peer assessment component is relatively high (20%), due to the emphasis we place on tutorials — the length of time spent, and the effort expended in learning the materials and sharing the knowledge. The students are asked to assess themselves and each person in the group in weeks 4, 8 and 13 (in a 13-week semester). When they are first asked to do this, students often want to give each other full marks, so their peers will give full marks in return and all will do well. When asked to justify such marks, it is often argued that all have been trying hard and contributing to group discussions (which may well be true). Accordingly, strict guidelines for the evaluation are provided at the start of the semester, and the tutors are given the final say in all such marks. It has been our experience that as the semester wears on, and some students skip tutorials due to illness, the

pressure of deadlines or work commitments, such peer evaluations become more realistic. This has proved to be a valid learning exercise for students at this stage of their undergraduate careers.

7 On-Line Aspects

One surprising aspect of this approach is that the materials developed for the tutorials turn out to be often (more or less) exactly what is required for on-line learning. In particular, as the tutorial material is intended as the main learning engine, it needs to be self-sufficient enough that the required syllabus is covered by the set of problems chosen. This is clearly independent of whether the problems are attempted by a group of six or so with a tutor present, or by a single student who is able to call on the services of a tutor by phone or email.

This approach can be most useful when lecturing expertise is not available. This is often the case for many on-line and international overseas systems, in which a pool of qualified tutors is available, but no-one with sufficient expertise or experience to give lectures in the subject. In such situations, a mixture of on-line resources, videos or reading material can be used to supplement the problem-based tutorials.

8 Conclusions and Further Developments

Good teaching practice always involves feedback and revision, and continual evaluation and improvement is to be expected. Nevertheless, we feel that the PBL approach to tutorials has been very successful and we foresee little change to this basic mechanism in the near future. This is not to say that the content of the problems won't vary, as they do now, to be relevant to the cohort of students enrolled. Based on comments that have been made over a number of years now, students generally find this more stimulating than "standard" tutorials, and, despite some initial reservations, generally support the self and peer assessment of their performance.

The on-line aspects were somewhat of a bonus, but are as yet not as fully explored. However, we expect that our experience with on-line versions of this subject will increase significantly in the near future.

References

[1] H. Barrows, *A Taxonomy of Problem-based Learning Methods*, Medical Education 20, 1986.

[2] D. Boud and Felleti, *The Challenge of Problem-based Learning*, Kogan Page, London, 1992.

[3] L. Cavedon, J. Harland and L. Padgham, *Problem Based Learning with Technological Support in an AI Subject: Description and Evaluation*, Proceedings of the Australian Computer Education Conference, July, 1997.

[4] T. Greening, *Scaffolding for Success in Problem-based Learning*, Medical Education Online Volume 3, Number 4, 1998. Available from `http://www.utmb.edu/meo`.

[5] T. Greening, J. Kay, J. Kingston, and K. Crawford, *Trialling a Problem-based Learning Approach to First Year Computer Science*, proceedings of the First Australasian Conference on Computer Science Education, July 1996.

[6] Rawnsley, Spaziani and Rangachari, *Evaluation in a Problem-based Course: What teachers want, what students think*, Proceedings of the International Conference on Problem-Based Learning in Higher Education, Linkoping, 1995.

[7] B. Duch, S. Gron, D. Allen, editors, *The Power of Problem-Based Learning, A Practical "How To" For Teaching Undergraduate Courses in Any Discipline*, Stylus Publishing, LLC (2001).

[8] D. J. Boud, (1985) *Problem-based learning in perspective*, in D. J. Boud (ed), Problem-based Learning in *Education for the Professions*, Issues of Teaching and Learning, Volume 2, Number 4, June 1996.