PAIRED PROGRAMMING FOR NON-COMPUTING STUDENTS

Gary Wills
IAM Group, ECS
University of Southampton
SO17 1BJ
gbw@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~gbw

Hugh Davis
Learning Technologies Group, ECS
University of Southampton
SO17 1BJ
hcd@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~hcd

Eric Cooke
IAM Group ECS
University of Southampton
SO17 1BJ
ecc@ecs.soton.ac.uk
http://www.ecs.soton.ac.uk/~ecc

ABSTRACT

In this paper, we describe a method used to teach non-computer science students how to program. We examine why students may not be engaging with the material and present an alternative method for conducting laboratory sessions. We allow students to self select their perceived level of expertise and use paired programming techniques. This has proved to be effective on several levels, engaging students and reducing demonstrator loading.

Keywords

self-selection, eXtreme, paired programming, agile development

1. INTRODUCTION

Learning to program when you are a computer scientist or an information technology student can be difficult. So, when it comes to learning to program for students of other disciplines, it can be particularly difficult especially when it occurs in your first semester at university and it is perceived to be a non-core function of your discipline. Not only are you trying to learn the principles and concepts of the subject, you are also having to overcome a lack of motivation and an unfamiliar environment.

Within the School of Electronics & Computer Science at the University of Southampton, analysis of entry grades and final degree classification indicates that a good A-level grade in mathematics is a good indicator that computer science students will do well at programming. So why are non-computer science students with good A-level grades, including mathematics, struggling with programming courses? Is the answer to be found in the way traditional programming courses are taught?

This paper examines an intervention into the teaching of programming to non-computer scientists. The intervention took the form of paired programming in laboratory sessions and student self selecting the category of programming expertise. This paper examines the background to the intervention, the implementation, and the observations resulting from the implementation. Finally, it presents some conclusions and suggested future work.

2. MODULE CONTEXT

The School offers three modules in procedural programming for non-computing students, all taught in C.

1. Programming principles
2. Modular programming
3. Data structures and algorithms

The Programming principles module is taught in the first semester of the first year and is a prerequisite for Modular programming. Modular programming is taught in the second semester of the first year and in turn is a prerequisite for the second year course on Data structures and algorithms.

The Programming principles module is taught to all first year, electronic, electrical and electro-mechanical engineers, together with students from other schools reading for a degree in science with computing (viz. physics, biology and chemistry). There is an average of 130 students registered on the module. The lectures are supported by 11 three-hour laboratory sessions (practical sessions). The module was taught for ten years prior to the intervention taking place.

The Modular programming module is taught only to first year electronics students, where the average number of students registered is 75. The lectures are supported by 4 three-hour laboratories which culminate in an assignment. This module was taught for four years prior to the intervention taking place.

In all the laboratory sessions, postgraduate demonstrators are used to help the students with any problems in understanding the exercises or code. The ratio of postgraduates to students is one postgraduate to twelve students. All postgraduates receive training in how to carry out the general

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

© 2004 LTSN Centre for Information and Computer Sciences

duties of being a demonstrator. The laboratories are supervised by a member of academic staff, who will also conduct specific training for the postgraduate demonstrators: how to mark the laboratory work, and the type and detail of help to give to students.

2.1 Student Survey

A member of academic staff supervised the four laboratory sessions on the Modular programming module. From the number and type of questions being asked by students during the first laboratory, it quickly became apparent to the member of staff that not all students were familiar with the prerequisite material.

Therefore during the second laboratory, a survey of the students' opinion of their ability to program was conducted. 45 students responded (68% of the class), of which 28 (62% of the respondents) rated themselves as having struggled or found it difficult to learn the basic C material from the first semester. These students were then asked two further questions:

- What extra tuition do you require? (see **Table 1** for results)
- How did you cope with the material last semester? (see Table 2 for results).

If there was to be extra tuition what areas would you like to see covered? (you can tick more than one)	Replies
How to design programs (algorithms, pseudo code)	8
Basic C syntax and functions	9
Advanced C (pointers, dynamic arrays)	23
More on modular design	12

Table 1 Response to question of student need for extra tutorial

How did you cope the first semester's course work? (you can tick more than one)	Replies
I used my colleague's code	0
I used my colleague's code as a template	17
I got a lot of help from my colleagues	19
I got a lot of help from the postgrads	20
I spent ages doing the exercises by myself	24
Other	11

Table 2 Response to question of how students found previous module's course work

The conclusion drawn from the survey and the observations from the laboratory session were that students were passing the first semester course, but without understanding what they are doing. The students were taking a surface approach to learning on the module [2]. Similar results were obtained in an anecdotal study by Jenkins, when he noticed that students at the end of a programming module "could not write even the simplest of programs" [10].

3. STRATEGY FOR IMPROVEMENT

From the information above, there were two areas in which the modules were improved:

- Course design
- Laboratories

3.1 Course Design

A review of the material from all three modules was undertaken, during which we noted the comments by Biggs [2] who, quoting Gardner, points out that coverage is the enemy of understanding [4]. Ramsden also makes the point that "we should strive to include less, but to ensure that students learn that smaller part properly" [14]. The course material was judged to be at the correct level for students entering each module with a pass from any prerequisites. Where there was overlapping content between modules, this was removed. For example, data structures as a topic was briefly covered in the Modular programming module and again in greater depth in the Data structures and algorithms module and was therefore removed from the Modular programming module.

While the module was taught in a traditional lecture style, it was felt that the laboratories and weekly programming exercises would provide the necessary engagement with the course material. Toohey [15] and Gibbs [5] suggest that laboratory and practical work is important for encouraging engagement with the course material. Gibbs and Habeshew point out that practical work in itself will not necessarily bring about deep learning unless students are given time to reflect [7], similar to Kolb's learning cycle [11].

3.2 Laboratories

The laboratories on the Programming principles module were redesigned to assess more accurately the student's ability to understand and write programs. Each laboratory has four stages.

1. Preparation. Read a relevant section or sections of the core text and make comments about this in their log book.
2. Find out how a program works. Load pre-written code, add comments to the code and describe the code dynamic operation in their log book.
3. Write a simple program. This requires only a small amount of code to be written, often a small extension to an existing program.
4. Write a more complex program. Normally the student would write the code from scratch.

Each laboratory is assessed using criteria marking; completing each stage of each laboratory gives the student an extra grade.

While Toohey extols the virtues of laboratory work in providing students with the opportunity to try out their new knowledge, get feedback, reflect and try again [15], Race points out that practical work is difficult to assess and very often what is assessed is the end product and not the process [13]. Race also noted that students do not like to have their performance observed.

The only reason that the Modular programming laboratories originally had any summative assessment (only 5% of the module mark for all four laboratories) is that it was felt that students would not turn up unless there were some marks attached. The students do have to understand the laboratories in order to be able to successfully complete the coursework (in effect we were marking the same work twice). Therefore the current cohort did not get any marks for the laboratories, but attendance and progress was still monitored.

Gibbs & Habeshaw point out that peer feedback of laboratory work can be quite effective in providing formative feedback [7]. In general, peer assessment can be a valuable tactic for students to internalise quality criteria and apply these to their own and other's work in order to improve quality, in a way that the tutors fail to do [6]. From the survey, we know that students are already struggling with some aspects of programming. However, to load them with the extra cognitive burden of learning to peer assess, albeit formatively, may be too much. Hughes also found some resistance to the idea of peer assessment when getting second year students to peer mark practical work [8].

The main concern raised from the survey was that after one and a half semesters of being taught programming, a large portion of the cohort was still finding it difficult to write relatively simple programs. An initial hypothesis was that students were having problems in laboratories because they lacked confidence in asking questions in large lecture groups.

To overcome this reluctance to ask questions during a lecture, buzz groups (answering questions in a group), and quizzes were introduced [7]. However, while students did participate in this learning and teaching activity, it was not followed by an improvement of student performance in the laboratory.

So, is there a way of combining the advantages of peer assessment, buzz groups (asking and answering questions in a group) and the learning of programming?

Originally, students were arbitrarily placed into laboratory groups of 12. Although they were placed in these groups, they were not required to work together in the laboratories. If students did "work together", they each had their own machine and each student carried out their bit by themselves. We know from Jenkins and Davy that students can be grouped quite accurately into groups of *Rocket Scientists*, *Averages*, *Strugglers*, and *Serious Strugglers* [9], and from Davis *et al* that students are pretty good at accurately self-selecting appropriate groups [3]. McDowell *et al* have shown that pair programming results in more student learning [12].

A strategy to improve the programming activity was to change these groups in order to pair the students of equivalent ability. The students would first self select appropriate categories (Not confident/ confident/very confident). Based on their self selection, they were placed in pairs of equal ability. It is important to put students of similar abilities and same genders together, to stop one of the pair taking over. Then during laboratories they work together; that is the two students work together on the programming task on the same machine. Students are encouraged to talk through the problem and solutions together.

This technique of programmers working in pairs has been popularised by the eXtreme Programming (XP) methodology [1]. Anecdotal evidence suggests that this methodology has led to programmers producing more robust code with fewer errors in a shorter time frame.

4. OBSERVATIONS FROM IMPLEMENTING THE STRATEGY

This strategy was first applied to the Modular programming course in the second semester in 2002/3. The initial observation was that the students spent over half the allocated time of the first laboratory to get used to

- the idea of sharing one machine between two students
- talking to each other about the problems and solutions.

The next observation was that the amount of work undertaken by the postgraduate demonstrators was significantly reduced, that is, they had to answer fewer questions. However the questions they did have to answer were generally of a higher level of difficulty than previously, so they spent the same amount of time in answering questions as in previous years.

The number of students that finished the exercises in the laboratories also increased. The normalised results over two years showed that the number completing the laboratory exercises rose from 47%

(2002) to 54% (2003). This shows that the students were getting to grips with more of the material.

On a practical note, to stop overcrowding, the laboratory's room size and number of groups remained the same. In each laboratory session, a pair could use only one machine for the programming while the other could only be used to display notes or look up the help information, while still providing the students with enough space to work in.

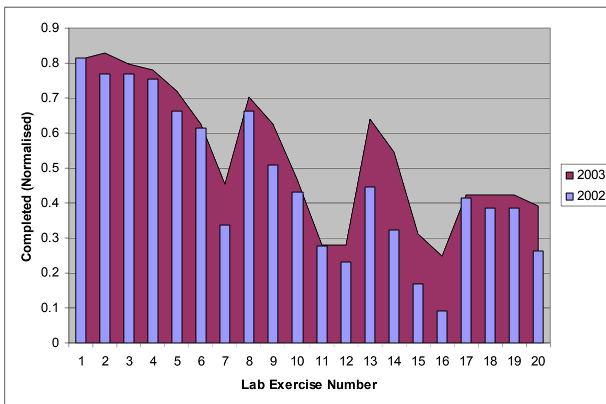

Figure 1 Graph showing completion of lab exercises

The strategy was then rolled out to the prerequisite module, Programming principles, in the latter half of 2003. The same cohort is currently using this method in the Modular programming module. What is of interest with this group is that more of the students are turning up to laboratory sessions even though the laboratories are no longer directly assessed. We believe that this could be partly a result of social pressure, "not letting down your laboratory partner" and also because they are enjoying the laboratories and finding them useful.

Similar observations were made regarding the reduction in the amount of work undertaken by the postgraduate demonstrators.

The significant difference, for the students, was not just in the improvement in course averages

- 62.9, (sd 15) in 2002
- 64.4, (sd 7.3) in 2003

but also in the reduction in the number failing to obtain a mark over 60%,

- 25% (17 students) in 2002
- 9% (10 students) in 2003.

This shows that more students are meeting the essential learning outcomes. With a strict marking scheme, the narrow spread of marks for 2003 indicates that more students were completing all ten laboratory sessions whilst working in pairs.

5. CONCLUSIONS AND FUTURE WORK

The paper has presented a solution for trying to engage non-computer scientists in the practical aspect of studying programming.

We believe this has gone some way to answering the question "does the method used in laboratory session hinder the students from fully engaging with the course material on learning to program?"

The intervention reported here is to let the students self select which category they are in with regard to their confidence in programming, and then use this to pair the students of equal confidence.

The result is that the work undertaken by the post-graduate demonstrators is reduced since the students learn from each other.

Grouping students for laboratory sessions is not new to our colleagues in other disciplines, for instance engineering or physics. It is common practice in those subjects to insist that students work in groups during laboratory sessions. The reason for this is not a matter of learning and teaching but of practicalities; that is, there is not enough (often expensive) equipment to go around. So in computing where there is enough equipment to have one student per machine, we moved to a situation that allowed individuals to write programs by themselves [12]. This was thought to be the correct solution since programming was seen as an individual task. McDowell points out that all non-trivial software projects require collaborative effort. Learning is also a social venture and by teaching programming as an individual task we lost the incidental learning that came from working in pairs and groups.

6. REFERENCES

[1] Beck K. (1999) *Extreme Programming explained: Embrace change*. Addison-Wesley.

[2] Biggs J. (1999) *Teaching for Quality Learning at University*. Open University Press.

[3] Davis H, Carr L, Cooke E, White S. (2001) *Managing Diversity: Experiences Teaching Programming Principles* in Proceedings of 2nd Annual LTSN-ICS Conference, 23-25 August, London.

[4] Gardner N. (1988) CTI and evaluation, *British Journal of Educational Technology* 19, 225-226.

[5] Gibbs G. (1992) Improving the quality of student learning through Course Design, in *Learning to Effect, Edited by Ronald Barnett*. The Society for Research into Higher Education & Open University Press.

[6] Gibbs G. (1999) Using Assessment Strategically to Change the way students Learn

in *Assessment matters in Higher Education, choosing and using diverse approaches* Edited by Sally Brown and Angela Glasner. The Society for Research into Higher Education & Open University Press.

[7] Gibbs G., Habeshew T. (1989) *Preparing to Teach; An introduction to effective teaching in higher education*. Cromwell Press.

[8] Hughes I.E. (1995) Peer assessment of practical reports. *Capability* (1) 39-43. On Line at: NCT Case Study Database www5.open.ac.uk/nct-database/case_studies/result.cfm?ID=61

[9] Jenkins T., Davy J. (2000) *Dealing With Diversity in Introductory Programming* in Proceedings of LTSN-ICS 1st Annual Conference, 23-25 August Heriot-Watt University, Edinburgh. On Line at: www.ics.ltsn.ac.uk/pub/conf2000/menu/menu.htm.

[10] Jenkins T. (2001) *Teaching Programming – A Journey from Teacher to Motivator* in Proceedings of 2nd Annual conference of the LSTN Centre for Information and Computer Science, London.

[11] Kolb, D.A. (1984) *Experiential Learning: Experience as the Source of Learning and Development*. Prentice-Hall Inc.

[12] McDowell C., Hanks B., Werner L. (2003) *Experimenting with pair programming in the classroom* in Proceedings of 8th Annual conference on Innovation and technology in computer science education, Thessaloniki, Greece ACM Press pp 60-64.

[13] Race P. (1996) *The Art of Assessing 2*. New Academic, Spring, pp3-6.

[14] Ramsden P. (1992) *Learning and Teaching in Higher Education*. Routledge/Falmer.

[15] Toohey S. (1999) *Designing Courses for Higher Education*. The Society for Research into Higher Education & Open University Press.