Easy and Effective Streaming for Introductory Programming Courses

Patricia Haden **Joy Gasson**

School of IT and
Electrotechnology
Otago Polytechnic
Dunedin, NZ
phaden@tekotago.ac.nz

The benefits of small-group hands-on teaching are well known in computer programming education. Unfortunately, with a heterogeneous student population, group members may work at very different speeds, and prefer very different levels of tutor interaction. This makes practical sessions cumbersome for the tutor and frustrating for the students. A more productive educational atmosphere can be achieved if students are placed in groups based on their abilities.

At Otago Polytechnic we recently trialled a simple method for streaming students into one of three laboratory groups for a first-year programming course. During the first class session, students were given a short quiz that emphasised their understanding of basic conceptual issues in program structure: flow of control, data operations and data assignment. On the basis of their mark on this quiz, they were assigned to one of three lab groups. Compared to previous instantiations of the course, this streaming improved the character of practical teaching for both students and tutors, demonstrating that a very simple assessment tool can be used to greatly enhance the educational experience. The strong correlation between performance on the streaming quiz and eventual course mark also highlights the importance of an understanding of basic structural principles to a student's ability to learn more complex programming concepts.

1. INTRODUCTION

The Bachelor of Information Technology at Otago Polytechnic is distinguished from other local IT programs by its focus on practical training and its commitment to individual education. To this end, the BIT emphasises laboratory practicals and places a strict limit on class size. Our practical teaching sessions are limited to 17 students, producing an extremely low student/teacher ratio, and allowing maximum one-on-one interaction between student and tutor.

Unfortunately, providing this level of individual attention is not always easy. Different students have different abilities and educational needs. This is particularly true of the BIT's student population, which ranges from 17-year-old school leavers to middle-aged tradesmen training for new careers. Tailoring instruction to accommodate such a wide range of ability, previous experience, learning style and educational goals can be nightmarish. This problem has been especially evident in our computer programming courses, where some students are experienced hobbyist programmers, and some are facing their first attempt at giving instructions to a computer. Mixing these students in the same small practical is good for neither group – the less experienced students are intimidated and the more experienced students are bored. It is also extremely difficult for a single tutor to provide optimal support to all students when they are moving at such different speeds.

In a recent offering of our second semester programming course – Introduction to Object-Oriented Programming – we decided to separate the students into streams based on their work pace and comfort with the computing environment. We canvassed our introductory programming tutors to get their impressions of what specific abilities distinguish strong and weak beginning programmers. Based on their experience, we identified three major programming concepts that "good" students have mastered, and "poor" students have not. In increasing order of difficulty, they are:

1. Understanding of the role of variables as data storage elements.

2. Understanding basic flow of control, specifically branching and looping.

3. Understanding the "step-by-step" nature of computer algorithms. That is, the need to specify everything the computer must do, in the correct order.

Note that there was no mention of any comparatively esoteric skills such as maths ability, logical thinking, verbal fluency, or any of the higher order intellectual functions that have sometimes been linked to successful learning of programming skills (Hartman, 1989). The impression of these experienced tutors is that successful programming students are those who have a firm grasp of the basic syntactic and structural mechanics of programming.

We constructed a short quiz whose items were designed to assess student ability on the simple metrics identified above. This quiz was administered to all students during the first teaching session of the Introduction to Object-Oriented Programming course. The quizzes were marked, and students were divided into upper, middle and lower groups based on their quiz result[1]. The students were assigned to laboratory streams based on this grouping. Students were told that different people have different levels of programming experience and that they would be assigned to the stream where we believed they would be most comfortable. It was made clear that this assignment was only a recommendation; students were not required to attend a particular stream if their schedules made it difficult to do so. However, the large majority of students voluntarily attended the recommended stream.

(Note that group sizes were not equal, as there were clear breaks in the distribution of quiz results between the top, middle and bottom groups.)

2. STREAMING QUIZ CONTENTS

Items for the streaming quiz were designed to assess a student's mastery of the basic programming concepts described above. An example problem for each category is given below:

1. Understanding of the role of variables as data storage elements.

In the code skeleton shown below, add the statements necessary to assign a value of 100 to the integer variable x, and a value of 25 to the 5th element in the array NumArray.

```
procedure AssignValues;
var
  x: integer;
  NumArray: array[1..10] of integer;
begin
end;
```

2. Understanding basic flow of control, specifically branching and looping.

After the following statements are executed, what is the value of the integer variable x?

```
x:= 10;
if x > 5 then
  x:= x*2
else
  x:= x - 5;
```

3. Understanding the "step-by-step" nature of computer algorithms

Assume that a value has been assigned to each of the ten elements of a global array of integers named NumArray.

a. Write a procedure that adds up all the elements in the array, and displays the sum.

b. Write a function that counts the number of times the numeral 5 occurs in the array, and returns the count.

The quiz comprised four simple questions, and was designed to be completed easily in the allotted time.

3. STREAMING QUIZ PERFORMANCE:

All students in the Introduction to Object-Oriented Programming course had recently completed, and passed, a 12-week course in Basic Pascal. All the material covered in the streaming quiz was taught in the Pascal course. One would therefore have expected uniformly high performance. Unfortunately, performance ranged from almost perfect to extremely poor. This is consistent with our previous experience teaching this course: although students come to the course with ostensibly the same amount of programming experience, there is considerable variation in actual skill level.

The pattern of performance on quiz items was as expected. Most students were able to do simple assignment, while some had difficulty with the syntax for accessing an array element. A number of students had problems with flow of control, and more than half of the students made errors when required to construct a full algorithm, as in the third problem

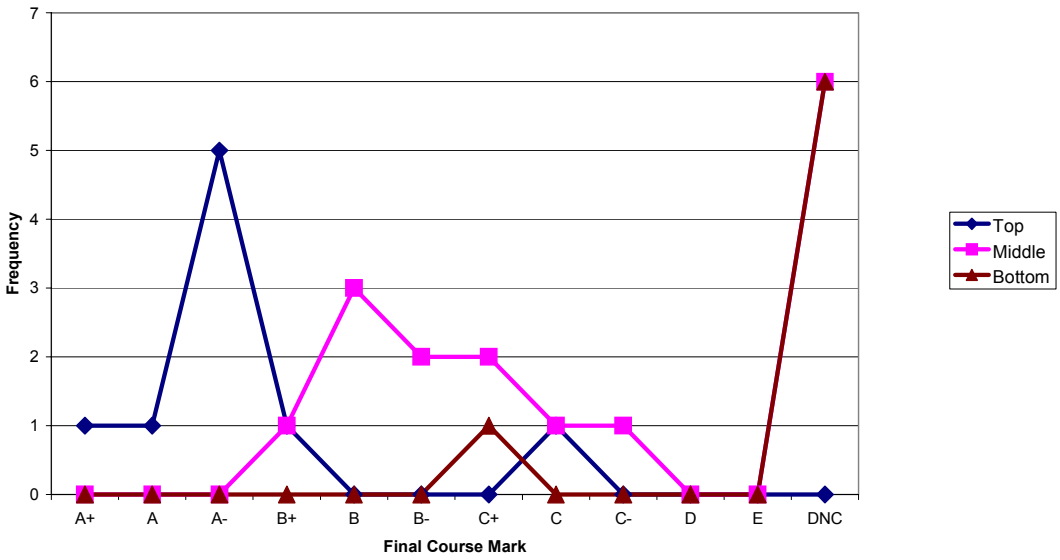

Figure 1: Distribution of Final Course Mark by Streaming Group

shown above. While discouraging, this result is in keeping with the poor performance of beginning programmers described in the literature (cf. Adams, 1996; Barr, Holden, Phillips & Greening,1999; Decker & Hirshfield, 1994).

4. SUBJECTIVE IMPACT OF STREAMING

Students were assigned to streams based on the results of the streaming quiz. All streams worked from the same laboratory manual, which contained content material and programming exercises for each of the 22 programming lab sessions.

It was intended that the stream containing the high performing students would move at a fast pace, with minimal tutor direction and maximal independent work. When necessary, the tutor for this stream would suggest extensions to the exercises in the laboratory manual to provide further challenge for the high-performing students. The stream containing the low performing students would move slowly, with more tutor-directed group work. The middle stream would move at an intermediate pace, with intermediate tutor support. Both tutors were experienced programming tutors and had used the laboratory materials in a previous offering of the course.

Of primary interest to us was the impact of streaming on the students' enjoyment of the laboratory experience. Previous offerings of the course had produced anecdotal evidence of dissatisfaction from both fast and slow students. The streaming approach eliminated virtually all these complaints. Most important, students in the slower streams reported satisfaction at being able to move at a comfortable, non-threatening pace, and receive extra tutor support.

Both tutors found the management of the streamed laboratory groups easier than the management of the more heterogeneous groups of past instantiations of the course. For example, in the slower moving sessions, the tutor had the option of demonstrating a programming exercise on her own computer, with the screen displayed to the students via data projector, and the whole class working through the exercise as a group. In previous classes where students were working at different paces, this synchrony was impossible.

5. RELATIONSHIP BETWEEN STREAMING ASSIGNMENT AND COURSE PERFORMANCE

A student who enters an introductory programming course as a strong beginning programmer is likely to do better in that course than a student who enters with weaker programming skills. While streaming can make the course experience more exciting for the strong student and less stressful for

the weaker student, it is unlikely to have a significant effect on the general ability of the student. This was demonstrated by an analysis of the relationship between initial stream assignment and final mark in the course. Figure 1 shows the distribution of final mark by streaming group.

As shown, students who were placed in the fast group by virtue of their streaming quiz performance generally earned marks in the A to B+ range. No student in this group failed the course. Students who were placed in the middle stream generally earned marks in the B+ to C- range. Six of the 16 students in this group failed the course. Sadly, only one of the seven students in the bottom streaming group was able to pass the course. Thus programming ability after only 12 weeks of introductory Pascal was an excellent predictor of performance in a second, more advanced programming course.

6. CONCLUSION

It is tempting to look at the pattern of marks described above and feel despair for the student who does not immediately show facility with programming. It seems that students who did not have a strong grasp of basic programming principles (as measured by the streaming quiz) after their first course were unable to perform to a high standard in their second course, even with the added tutor support provided to the middle and bottom stream groups. While there is no doubt a confounding influence of general student ability, it seems as though with programming, if you don't "get it" right away, you may never do so.

We prefer to look at these results as a guide to the proper emphasis in our early programming courses. High performance on the streaming quiz did not require any remarkable intellectual prowess, mathematical skill or logical analysis. It simply required that the student have mastery of the basic programming principles of assignment, flow of control and algorithmic thinking. Tutors generally find first programming courses, with their emphasis on syntax and simple algorithmic thinking, less interesting to teach than more advanced courses that incorporate high level design concepts and creative algorithmic approaches. However, the results discussed here demonstrate that the importance of these basic concepts must not be underestimated, and their teaching must not be neglected. The student who is short-changed in acquisition of basic programming logic is at a disadvantage, while the student who truly owns the logical foundations of programming will be poised to excel in whatever programming environment he or she may encounter in the future.

References

Adams, J., Object-Centered Design: A Five-Phase Introduction To Object-Oriented Programming In CS1-2, SIGCSE 1996

Barr,M., Holden,S., Phillips,., Greening,T., An Exploration of Novice Programming Errors in an Object-Oriented Environment, SIGSCE Bulletin, pp. 42-46, 31,4,December, 1999.

Decker, R. & Hirshfield, S. The Top 10 Reasons Why Object-Oriented Programming Can't Be Taught in CS!, SIGCSE 94

Hartman, J. D. Writing To Learn And Communicate In A Data Structures Course. ACM, 2, 32-36, 1989.