UNIVERSITY GRANTS COMMITTEE

Conference on Teaching Computing held at the University of Bristol on 28th March 1972

PROCEEDINGS

WHAT SHOULD WE BE TEACHING?

by
Professor S Michaelson
University of Edinburgh

Before the nature of courses can be discussed, there must be some agreement about the objectives of the courses. That agreement must cover the type of student we intend to teach, and what the courses are intended to convey to them. The content of the courses can partially be deduced from these decisions, but will be influenced by the resources available for the teaching.

Let us start by considering the sort of students for whom universities should offer courses. I think that we can start by distinguishing 2 categories:-

a. Those who wish to use computers as a tool in their studies, and

b. those who wish to be educated about computers in a general sense.

School teachers who wish to teach computer science in schools should be trained as specialist computer scientists, just as physics and mathematics teachers are (this is one good reason for offering joint honours courses including computer science, for they provide the subject training needed for teachers who may have to teach two or more subjects, or across subject boundaries). In addition, all school teachers should have the opportunity to acquire a general education about computers, and so should receive courses aimed at category b., at the very least.

Consideration of much computer use shows how often apparent skill is a whited sepulchre, hiding muddle and the squandering of valuable resources: hiding, too, abysmal ignorance of generalities about computers and their uses in society and their effects upon society. We need to raise the standards of technical users of computers in two ways. We should give them a better general understanding of computers, and should encourage them to develop real skill in computer use, based on an understanding of the structure of what they are trying to do. Experience shows that the course of 8 lectures on Fortran, so beloved of our masters, is thoroughly unsatisfactory for teaching technical skill. Far too many programmers have been trained in this sort of way, and every computer centre can tell stories of the wasteful consequences.

What do we want to convey to students, under the head of general education? We need to dispel the mystery which surrounds computers and their users. The 'inevitability of computer use' must be shown for the sham it is. Naive belief in the answers produced by computers is a social evil which should be eradicated, and we are responsible for its persistence among those who have been educated at universities. We should encourage a spirit of informed, health scepticism which does not accept without question assertions of the truth of this or that result, or of the necessity for the use of computers in this or that way.

There does not seem to be any way in which most people can acquire a feel for computing without using computers. How should these students use computers? Clearly they should use them as tools in solving problems, and these problems should impress them as realistic. This requires us to provide them with languages and packages[1] that will enable them to perceive what they are doing as they follow

1. It should be pointed out that most of the currently available packages have control languages that offend all the canons of good design and we need to provide packages which are controlled as clearly as the flow through a program is controlled.

the thorny path from a problem to its solution. We now understand how important it is that programs should have clearly visible structures, so that their behaviour can be understood. This requires that students should be prevented from writing knotted programs. During the last few years, Wirth, Nauer, Dijkstra and others, have shown how a few <u>really</u> simple control facilities can replace the spurious simplicity of labels and jumps. If we ally this with the "principal of minimum detail" we are well on the way to permitting the students to see the forest in spite of the trees. The approach to a problem by "peeling off the skins of the onion", working at the least detailed level at which one has an adequate tool, has been well exemplified by Dijkstra. Not only a language must be provided, but also packages of complex operators upon complex entities, so that students who do not need to study the detail can solve problems by using a "machine" which handles things at a level appropriate to them.

Once students feel that they can master the computer even if it is merely one with such complex operands as dictionaries, and such complex operands as "find this word in that dictionary", we can talk to them about commercial and industrial applications and expect them to understand how "garbage in" leads to "garbage out". We can explain how it is that people lose sight of the potentially dangerous effects and the inefficiencies of computer use, in the joys of being able to do complex things easily. We can warn them against exaggerated claims and expect them to follow us when we tell them case histories of complex systems.

Every technical user also needs the same sort of understanding of how one goes about solving problems, of how one reduces the effort required of the problem solver by teaching him to program clearly. Every technical user also needs to have some basis for judgement of contending methods of solution of his problems. We should give technical users the same sort of introduction as we offer for general education. This is satisfactory if technical users can continue to "peel the skins off machines" until they reach the level of detail appropriate to their technical use. This will encourage them to write clearly structured programs at a fairly detailed level, thus raising the general standard of technical skill and efficiency considerably above the present level. This approach has already been used for some time in teaching specialists about compilers, operating systems and interrupt handling and so on. It is only recently that we have begun to use it for introductory teaching. Our students in Edinburgh have found it easy to learn assembly languages and common languages, such as Fortran and Cobol, after becoming able to program in IMP (a block-structured language), and there is no reason that students in general, once they have been taught to program decently in a language well adapted to teaching basic principles, should find difficulty in picking up other programming languages. The usual introductory courses, in Cobol, Fortran, PL/1, and so on, are not an adequate substitute for proper education using a well-designed pedagogic language.

An approach of this sort starts the user off on the right foot. Writing well-structured programs becomes the natural thing to do. Programming is simplified. It becomes easy to modify programs, and to keep them correct. The programmer becomes more willing and more able to plan solution processes to allow for changes in the requirements of the users, and a general economy of human effort results. At the same time, it is easier to improve the economy of use of the computer, because it is easier to pick out the parts that need improvement, and to improve them.

The effort required of the student in the general course is not negligible. The student has to learn a new language, in which to express his ideas. He has to absorb many new concepts - flow of control, storage, sequencing of events - to adopt a new approach to problem solving, more reasoned than before, more explicit, without appeals to his audience's intuition - to deal explicitly with more complex

processes than before, and, above all, actually get things right. We find that this stage occupies about one third of a student's time for a term.

To try to do this in a non-credit course in unreasonable, and many universities will have to modify their course credit structure somewhat before they can offer this sort of thing to all their students. This also demands a lot of teaching effort, especially if the social problems are to have worthwhile discussion, and it is not the sort of teaching that can be done by hiring final year mathematics students. It needs people with a fairly mature understanding.

It also needs good computing facilities to maintain the students' will to work. Rapid turn-around sessions and remote job entry terminals are very useful, but the best tool is a good multi-access system - reliable, fast, with video-terminals, and easy provision of hard copy.

The specimens used in the teaching need to vary with the students' interests. The depth of detail that is exposed must vary with the students' need. The rate at which students can absorb this sort of material appears to vary greatly from student to student. A large range of differing courses is needed. What better way of providing it is there than the use of a computer as a teaching aid? Clearly this is the time for an 'imaginative experiment' to be carried out - for the UGC to fund a few universities on a large enough scale to enable them to use a good multi-access system as a tool in large scale introductory teaching.