THE TEACHING OF COMPUTING

Fred Gruenberger

October 1964
THE TEACHING OF COMPUTING

Fred Gruenberger*

The RAND Corporation, Santa Monica, California

There are now hundreds of courses in our colleges--and high schools--that offer an introduction to computing. Some of these are rather elementary, amounting to instant Fortran, or the mechanics of coding for a specific machine. The better ones teach the art of computing and problem-solving, or try to. As always, the good ones are in short supply because of the lack of good teachers. Our field is no different from others in this respect--the job calls for people who understand the whole field, who know how to teach, and who enjoy teaching.

The stock of such people might be increased if more people were aware of how enjoyable this job can be. It is significantly different from the teaching of any other subject and a list of some of the reasons for this difference might be helpful.

1. There can be personal verification. Everything the instructor claims for the machine or the language or the system can be personally checked out by each student. Nothing need be hidden or secret; in fact, in the final analysis, nothing can be kept secret.

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.
2. The answer to almost every question of the form "What happens if...?" can be, "Why don't you try it and find out?" It may not be efficient to do this constantly, but it is always possible. The contrast to imaginative questions in, say, a chemistry course is quite marked. A certain amount of such exploration in a computing course is desirable.

3. The real teacher is the computer. In computing, theory and practice interact closely. If console debugging is permitted, it is there that the student will light up and say "Oh! That's what he meant." That "Oh!" is what every teacher strives for. It occurs in computing more often than in other subjects.

4. There is always a better way. Another expression that a teacher of computing waits to hear is, "You're doing it all wrong!" Hopefully, some student, sooner or later, will notice that what is offered as the standard way of doing something is not as good as it might be. The fresh pair of eyes has detected a better way of accomplishing something, and the owner says so. Frequently the student's "better way" is only a rediscovery of an old, abandoned technique, but at least it has been independently re-invented—the student has been inspired to do some independent thinking. This procedure could also take place in, say, a physics course. But it is more likely in a good computing course.

5. The student of computing can be brought to the boundaries of the art very quickly and may even advance beyond those boundaries. It is becoming commonplace for today's students to write new compilers, for example.
There is a corollary to this point. The beginning student of computing can be involved with real problems very early in his course. He must go through many small finger exercises, to be sure, but even these can be made part of real work.

6. Computing is still a dynamic subject. A week cannot pass without some news in the industry of a new machine, a new technique, or a new book. Last semester's lecture notes just won't do. Even last semester's examples tend to look fatuous; there are better ones now.

The dynamic nature of computing gives the student a sense of immediacy and intimacy equalled only by a class in space flight. The current issues of journals can and should be part of the course. The beginner can enjoy being current, especially while reading the advertising pages.

7. The computing field has been explored by some very bright people, and many of the everyday tricks of our trade are really brilliant inventions. A list of such things includes indirect addressing, index registers, closed subroutines, and delay-line storage devices, to name just a few. To the beginner, all such inventions are neat and clever. He will enjoy learning about them, particularly if the teacher creates a need for them first.

As in all teaching, these topics should be introduced with flair and analogy. The search for homely examples is endless. Consider this way of explaining indirect addressing:

My friend Reynold Johnson helped entertain Premier Khrushchev on his U.S. visit, and shook his hand. At that point, Khrushchev represented zero-level-addressing to Rey: the object sought was right at hand, so
to speak. Now I shake Rey's hand; Khrushchev is sort of first-level-addressed to me. If my student now shakes my hand, he is second-level-addressing from Khrushchev, and so on. Within fourth-level-addressing of this kind, one can make contact with most living persons. (My students can reach von Neumann in first-level-addressing; in theory, we could all be about fifth-level-addressing from Lincoln.)

8. In any other subject, we try to teach only sophisticated methods. Frequently, in computing, we can go the other way. We can teach crowbar methods and brute-force techniques. The straightforward, stupid way to the answer can be the best way for the beginner.

On the other hand, there is unparalleled opportunity in a computing course to demonstrate the principle that brains beat brawn—that a modicum of thinking can replace miles of coding and hours of machine time. The literature contains many case studies, but one from personal experience is typical.

In sifting out the first six million prime numbers in 1959, C. L. Baker and I used a sieve that we thought was fairly efficient. The run, on the IBM 704 computer, took 120 hours—part of which, to be sure, shouldn't be counted, since we punched the results. If 12 per cent of the time was then wasted, the run still exceeded 100 hours, and over 90 per cent of that time was spent with just two instructions.

In 1963, the primes were again sifted, this time using a 7090 and a much better algorithm: the run time for computing was something like 21 minutes.

However, these two opposing philosophies (the crowbar approach vs. brains-beat-brawn) are not inconsistent. The first attack on a problem should use the crowbar approach,
unless the slick, sophisticated method leaps to mind right away. It is only after some cases are run that the programmer has the feel of the problem and can then attack it with efficiency.

9. A shining characteristic of the computing business is the high degree of cooperation shown among users. Individual programmers have learned to cooperate, to the extent that program libraries spring up for every machine type. Installations cooperate with each other, if only for backup in the event of machine failure.

This cooperation is vital to our industry. It can be taught to students of computing; they should learn, from the first day, to depend on the cooperation of others and to offer their own cooperation to others.

10. Computing revolves around communication and in a computing course many forms of communication can and should be taught. Basically, communication with the computer must be stressed, but along the way we can show how to communicate between the student and teacher, and among the students.

11. There are many excellent films available for classroom use. Many of them were created as sales films, but what is teaching but a combination of dramatics and salesmanship? Honeywell's film on parallel processing is an example; in ten minutes, the whole concept becomes clear.

None of the differences listed above is significant in itself, but the aggregate makes the teaching of computing an interesting and satisfying experience. There must have been much the same feeling in teaching television in 1940. Perhaps some day we will know just how to teach computing--and life will be duller. In the meantime, our profession needs more well-taught courses from top-notch people.