COMPUTERS IN EDUCATION, K. Duncan and D. Harris (eds.)
Elsevier Science Publishers B.V. (North-Holland)
© IFIP, 1985

Macintosh: Rethinking computer education for engineering students

Andrew U. Frank

Department of Civil Engineering
University of Maine at Orono, Orono, Maine 04469

Computer education for engineering students is an open problem. The advent of a new type of affordable hardware, the Apple Macintosh, which introduced a new concept for the user-machine interface opens new possibilities for computer education. This paper discusses (1) the possible goals of computer education for engineering students, (2) a new concept for introducing computers, and (3) how the Macintosh is being used to achieve those goals at the University of Maine.

1. Introduction

Education in computer usage, i.e. how to use a computer to best advantage to solve applied problems, has become a problem in many science and engineering departments. Most students take an introductory programming course early in the curriculum. Because of the unforgiving nature of the mainframe systems usually employed, they spend enormous amounts of time on trivial assignments and little is left for more realistic and complex problems. When they are later asked to use computers for solving technical or scientific assignments, they are ill prepared. As a result, computers are not used to full advantage in our present educational system.

Modern personal computers, which are easier to operate, allow students to concentrate more on the essential goals, and to minimize extraneous problems related to idiosyncrasies in complex operating systems. The new Apple Macintosh is perhaps the most outstanding example of an easy to use machine. The Macintosh incorporates for the first time the concept of 'direct manipulation in a machine in the price range of a personal computer. It uses novel hardware, (i.e. a powerful16-bit microprocessor and a high resolution, bit mapped graphics screen) combined with innovative operating system software to provide the first low cost system with an effective 'visual interface'. Operations inside the computer are made graphically visible for the user which does not only help the experienced user, but dramatically reduces the beginning students problem of understanding and using the system [9].

This paper describes a serie of two courses offered by the Civil Engineering Department of the University of Maine at Orono, as a new approach to the challenge of more effective computer education for engineering students. The department is a typical civil engineering department with approximately 250 students. It is part of the University of Maine at Orono, a smaller land grant university with about 11,000 students. The course discussed in this paper is considered to be an experiment within the College of Engineering and Science. If the course is successful, we expect it to be duplicated in other engineering departments. Moreover, we feel that these ideas can be effectively applied to computer education in other fields as well. We advocate that others critically examine their present offerings in this area, and develop similar courses to better respond to the needs of their students.

2. Goals of education in computer usage

Everybody asks for computer literacy. However, no generally accepted definition exists. The following is a list of what we see as important goals for computer education for engineering students. Computer education should enable students not only to use today's computer efficiently, but also to understand the basic principles involved so they will be able to stay abreast of future changes. It is thus obvious that an effective course must do more than teach a computer language. The course must concentrate on the general principles which are unlikely to change rapidly, and use present systems only to illustrate these principles in computers. Students must understand which parts of the computer environment they will find similar on other machines, and which are particular for the machine used for instruction.

2.1 Understanding computers - It seems necessary that students understand the hardware components of today's computers, their functions, limitations and advantages. It is equally important to know the basic principles of software; not only the operating system, but also the compiler, linker, editor and other utilities.

2.2 Information and data - To understand how computers may be used to solve problems is facilitated by understanding the characters of information and data. Such knowledge may prevent the widespread confusion about what computers can (easily) do and what operations are (still) reserved for human beings.

2.3 Communication - Communication of ideas and exchange of data are becoming more and more important. The principles that apply both to communication between machines (networks), and to the effective use of output for human understanding (print, screen, graphics, voice) must be understood.

2.4 Programming - All engineering students should learn at least one programming language and become familiar with it. We do not assume that most practicing engineers will write their own programs; instead, we foresee general use of software packages, produced by specialsts. Nevertheless, teaching a programming language seems essential in order to provide the student with a

framework to explore and apply the more abstract topics mentioned thus far. Additionally, students must be able to apply programming in order to solve problems put forward in later courses.
We do not agree with the approach taken in [4] which provides the students with finished programs and discusses the mathematical principles only; it appears too easy for students to gloss over problems, and they are not exposed to the task of converting a mathematical idea for a solution to a program.

3. Insufficience of 'Introductory Programming' courses

Judging from the available textbooks, the average computer course today is an introductory course in the use of a programming language; usually FORTRAN in engineering programs. This approach is insufficient for several reasons:

3.1 Batch orientation - Many books still assume that the student will use punched cards even though punched cards are now seldom used in practice and will disappear completely in the immediate future. All aspects of interactive programming are left out in most books (a noteable exception is [2]). This makes it difficult for the student to understand and assess the interactive programs she will later use.

3.2 Concentration on 'programming language' - The course introduces students to the use of a programming language alone. Most often the use of the operating system and the utilities are explained only as far as absolutely necessary for the development of small programs. Students are not explicitely told how to use the computer for other problems, nor are the principles of the operating system explained to them. This greatly hinders future use of the computer in advanced courses.

3.3 Examples unrelated to engineering - Examples in general text books are unspecific in order to be understandable to students from all fields. They are often simplistic and do not relate to students of engineering. Furthermore, the examples do not prepare the student for the type of programs she will later see when applied to her field. Programming has become so broad an activity that very different knowledge is necessary to work in a specific field of application. This can not be achieved by a general course.

4. Integration of computer usage in the curriculum

If we want to equip our students with a sound understanding of the best use of computers in their profession, we must integrate computer usage in their curriculum. An independent course little related to the rest of the curriculum can not achieve this as it remains an unconnected topic in the students experience (and much too often an unpleasant one).

Discussions in our college have shown that students can achieve proper experience with computers only if each student takes at least one course with a strong computer usage component **each semester**. This means that we have to follow the introductory course (in first or second year) with applied courses (e.g. an engineering design course) that include some use of computers. Such a component may be built into most practical or design courses without taking time away from the principle subject. Quite the contrary, we experience that many topics may be studied in more depth if computer simulation is available or if more design alternatives can be explored.

5. Modern personal computers and their use in teaching

If we analyse what is difficult for students when they deal with the traditional mainframe computer, we can better assess the improvements possible with the newest personal computers.

5.1 Invisibility: computers handle objects like files, records, etc, which are invisible to the user. It is difficult for a beginner to keep track of the multitude of objects of different kinds she operates on with commands. Eventually, users understand and develop a feeling for these things - as all of us have done - but this is very time consuming and entails many 'trial and error' situations and disappointments.

5.2 Inconsistent interfaces: a specific action - say erase an object - is expressed in very different ways, depending on the class of the object and the situation (not only a different command, but also different syntax).
For example, compare the difference between how you delete a file and how you delete a line in a file in the system you are familiar with. The problem is aggravated as each utility and application program presents another type of interface. Again, users may eventually become proficient in their use, but the program behaviour is baffling to the beginner.

5.3 Overly complex systems: most general purpose multi-user mainframe computers now in use provide extremely versatile command languages. However, their functionality is much larger than students normally need, and students have difficulty selecting the subset they need. They may encounter problems if they inadvertently use an unfamiliar mode. Typically manuals are several large binders with over 500 pages each, written by specialists for specialists - a language completely incomprehensible to the beginner, and unfortunately often specific to the manufacturer.

5.4 Difficult to get started with: starting to use a general-purpose mainframe is made additionally difficult by the provisions in their operating system for tailoring them to the users need. Very often, beginners have to deal with the "naked" operating system until, much later, they have created their set of 'macros' to abbreviate command sequences often used.

We may conclude that a machine suitable for teaching computer usage must be simple to use for the beginners. To judge a machine to this end may be difficult as all experienced computer users have developed considerable skills to deal with even the most intricate machines. The following properties seem to be important:

- visible interface: the user's actions should cause visible reactions on pictures of the objects manipulated (also called 'direct manipulation').
- consistent interface: a similar operation should be performed with similar commands and similar syntax.
- a limited operation system with the necessary functionality, but not providing too many additional features which only confuse beginners.

- let the user select or adapt a proposed command in lieu of asking him to produce the command [13].

Several attempts to realize these goals have been made previously. Notable in the past are the Waterloo Environment to reduce complexity of the IBM operating systems on the mainframe side. Single user workstations have been produced by Xerox (Star), Apollo [3] and Apple (Lisa and Macintosh). The last one offers the necessary features for the lowest price (around $2000 per personal computer).

Selection of a computer for use in a computer course is similar to other selections of computers: decide which qualities are essential and select the one that has them. The danger lies in being tempted to buy a machine with more features than necesary. Such features are never free, they may perhaps not increase the price, but they will invariably increase the complexity of the user interface and make the machine harder to use.

6. Guidelines for the new course sequence "computer usage"

This chapter will discuss the main assumptions and guidelines we use in the course and present the outline of its contents.

6.1 Reduce complexity
We assume that complexity of the task is the major impedient in any programming course. Complexity can be reduced mainly by introducing new topics one at a time only, and have the student exercise this topic before anything new is added [6].

6.2 Reduce problems not related to the present goal
Beginning programmers have to deal with a great deal of detail, especially in order to gain access to the mainframe, control the invisible environment there, edit, compile, link and run their programs. Single user systems can be made simpler as they serve for a smaller selection of tasks. Personal computers, like the Apple Macintosh with its visible interface, further reduce the amount of knowledge and experience a beginning user needs in order to write a program.

6.3 Fast reaction of the system
The time lapse between a user's action and the system's reaction, for example during debugging a program, are critical. If response time is extremely fast - say in less than ten seconds between editing an error and seeing the change in the output from the running program - program development for beginners is much faster. Not only does each step take less time, but the involvment with the problem is not interupted and facts are remembered in the learner's short time memory. We believe this fast response enhances learning by helping the user to assimilate patterns in a way similar to the assimilation of typical patterns of behaviour in ordinary life.

6.4 Reduce technical details
The Pascal Interpreter for the Apple Macintosh (made by Think Technology Inc.) is an integrated system, providing a language specific editor together with advanced debugging tools in one package with a uniform interface. The editor checks the syntax of the program during input and signals errors immediately; it also indents the program according to standard rules (prettyprinting)
This should especially help beginners who often have problems with some of the minute details of Pascal syntax (especially semicolons and begin - end pairs). Éliminating these hinderances will allow increased concentration on the essentials of program design.

6.5 Include interactive programming
Design of interactive programs is cumbersome on most mainframes, at least for the beginner. There are always several system idiosyncrasies to be considered so textbooks usually do not discuss these areas. The input procedures of the Apple Macintosh are quite simple to use (lazy I/O).

6.6 Include graphic output
The high resolution screen built in the Apple Macintosh, together with its high performance bit mapped graphic routines allow integration of graphic output without adding much complexity. Graphics can help enormously to make the user visualize the dynamic behaviour of programs and lends itself to many rewarding simple programs (see the publications of the LOGO group [7] [10]).

7. Teaching methods

7.1 Program plans
We assume that the problem of programming does not lie primarily with the syntax and vocabulary of the chosen programming language, but in the difficulty of devising a plan to solve a problem. Experienced programmers have accumulated a number of plans that they have used previously. When they have to solve a new problem, they choose from a stock of basic ideas, which are then adapted to the specific situation (often by copying the previous program and editing: changing is easier than inventing [13]). Beginners, however, are asked to create solutions from the void - an immensely more difficult task.

7.2 Programs are for reading
Programs must be considered primarily as accurate and formal descriptions of an algorithm, and should be readable by humans. The additional benefit of programming languages that algorithms can be executed by a computer should not influence the design and notation [5].

7.3 Reading assignments
Reading one's own programs as well as programs written by others must be an integral part of teaching programming.

7.4 No initial optimization
Algorithms should be described as clearly as possible and at first no effort should be spent on optimizing (neither for run time nor for storage utilization). Optimization invariably makes a program more difficult to understand. This is not to advocate inefficient algorithms, but to avoid optimization tricks that obscure the meaning of a program. If a program later is running too slowly, program transformations can be used to speed it up.

7.5 Programming style
Students should not only learn to write a program that produces the correct results, but to compose a well written program. This is not an end in itself, but experience shows that clumsy programs seldom work correctly, are difficult to debug and impossible to change later. It is a disservice to the learner not to insist on programming style.

7.6 Reasoning about programs
The mathematically founded methods for reasoning about programs, as used in formal proofs [5] [15] [6], must be introduced early to the students.

7.7 Modularization
Even small problems are too big for the beginner, and students should learn from the very beginning how to split a problem in to smaller problems which are easier to solve. Our personal experience is more with the data abstraction technique (also called abstract data types or initial algebra) [11,12] than with procedural abstraction, and we feel that it is important to understand and apply these methods even in cases that seem too simple. It is too late to learn new techniques when complex problems are calling for them. Personal experience with writing large and complex programs has shown that routines that are very small and embody only one idea are easy to write and seldom contain an error (by our standards, routines with more than 10 executable statements are considered long!).

7.8 Functional programming style
Small routines designed using the principles of data abstraction lend themselves to being written as functions. Functional programming languages and functional programming style were quite successful in the past and - at least applied with judgement - seem to lead to programs easy to understand. Moreover, the idea of a function is well established with university students and this allows us to exploi previous knowledge.

8. The outline of the course

The goal of the course is to have students understand computers and become familiar with their use. The first goal is achieved through the later - 'learning by doing' is the catchword. Therefore a great number of small assignments are given to the students, two each week and - if one is done after the other - each taking about two to three hours to fullfil. Many of them will be assignments to change an existing program. Some assignments may form a sequence, where the program written in the previous assignment is expanded over several steps. This should provide the students with experience in building programs using stepwise refinement [15].

The books used for the courses are the following:
- Systematic Programming' by N. Wirth[15], which is one of the few introductory texts, which discuss programming in general and are not concerned only with syntax and vocabulary. It is also the only book I found that introduces formal reasoning about programs.
- 'Numerical Methods Using Pascal' by L.Atkinson and P. Harley [1] for the numerical part.

We also recommend the following addional texts to students who are interested in further reading:
- 'Mac: The Apple Macintosh Book' by C. Lu[9] for more details about hardware.
- An introductory text in Pascal about non numeric programming if they feel they need it.

A lab is set up with 8 Apple Macintosh for exclusive use by students of this course, yielding a 1 in 8 ratio between students and machines. The lab is open every weekday from 9 a.m. till 9 p.m. We encourage the use of the Macintosh for work related to other courses.

The course is divided in three parts:

8.1 Familiarize students with the Macintosh
The first week is used to familiarize the students with the interface of the operating system and the editor to write short documents. We assume that students will, on their own initiative, also explore the graphics editor (MacPaint). The consistency of the interface across all programs should make this period short; nevertheless, two assignments will insure that all students get the necessary practice before the next step. Lectures will be used to discuss the hardware parts and the first presentation of the task of the operating systems.

8.2 Introduction to programming
Several weeks (three to five) will be devoted to the general idea of programming. Students will be presented with complete, running programs to read and change. Only a limited part of the Pascal vocabulary will be used, not with the intention of subsetting, but rather by excluding all but the most obvious constructs (including functions with value parameters).
We start with the 'triangle example' from [7] and motivate the introduction of procedures and functions as abstractions (abbreviations) of repeatedly used task. This leads in a natural way to recursion, and we did not observe any problems with students understanding this.
The next group of examples are operations on number systems (integer, rational numbers, complex numbers and ultimately polynoms). This provides us with mathematically clean algebraic examples for abstract data types. The idea can then be expanded and applied to less 'clean' practical applications. Assignments in this phase are program reading and small changes as well as writing larger sets of subroutines. This period concentrates theoretically on the idea of abstraction and layered architecture of systems.

8.3 Programming language Pascal
When the students have gained some experience with programs, the time is ripe to introduce them to the details of the Pascal language in a systematic way. We present the language in a logical order and explain details which either have not yet been covered or have been covered incompletely. Here the stress is on showing the students the systematic construction behind programming languages in general, and Pascal as a specific example. Programming assignments will be split between reading and writing, with examples preferably showing some building blocks often used (e.g. filling and searching arrays). All constructs of Pascal but pointers are introduced in the first semester.

8.4 Programming for engineering application
The second half of the first semester and the first half of the second semester will be devoted to building a library of modules useful for building application programs in engineering and science. Topics to be treated include:
-handling polynomials, including derivatives and integration, finding zeros and maxima/minima, and graphical output
-dealing with arbitrary functions, including finding zeros, differentiation and integration, using numerical methods
-advanced matrix operations, i.e. solution to systems of linear equations, inversion of matrices, eigenvalue and eigenvectors
-simple cases of systems of (non linear) equations
-introduction to numerical solution of differential equations.

Numerical methods take a large share of time, but we will also treat at least one example of calculations related to networks (either flow or critical path) in order to introduce the techniques applicable in this area [14]. This is also the occasion to explain the use of pointer variables in Pascal.

8.5 Transition to the mainframe and FORTRAN
Starting in the second semester, students are gradually introduced to use the mainframe (IBM under VM/CMS). The goal is to enable the users to decide for themselves when to use what machine, and to become aware of the relative advantages of each. An introduction to FORTRAN is also included which should enable students to use and possibly adapt existing programs written in FORTRAN. According to [8] students do not encounter problems when moving to the mainframe, and the skills they learned first are transported easily.

9. General changes in the curriculum

A computer usage course with the contents explained above can show mathematics in a new, more applied light. During development of the course, we looked carefully at the use of mathematics in engineering disciplines, especially civil engineering, and compared it with the present contents of the compulsory mathematics courses.

9.1 Calculus and differential equation
Calculus is the foundation of modern mechanics, and therefore of prime importance for most engineering sciences. Nevertheless, it should be asked if the present nearly exclusive concentration on calculus in our mathematics introduction is justified. The present contents often seems to be more directed towards mathematical proofs and results of theoretical value but very seldom motivated by the way calculus is applied in engineering sciences. In my opinion, the reason for this is that most applications of calculus in engineering lead to problems which can only be dealt with using numerical methods (eg. most integrals defy formal integration, most differential equations of interest to engineers can only be solved approximatively). However, numerical methods have become accessible only now, using computers and programming.

9.2 Algebra
Data abstraction - a very important method in specifying and designing computer programs - is directly connected to abstract algebra. Understanding the abstract properties of algebra and being able to look at problems using these methods are helpful.
The basic concepts of set theory should be available to all students. Such concepts are applied in many cases of programming and provide powerful tools to analyze the operations of programs (and some programming languages include sets as basic data types and provide the necessary operations).

Boolean algebra (including predicate calculus) must be introduced to students to enable them to reason about conditional statements in programs.

Further, the generally used number systems (integers, rational numbers, reals and complex numbers) should be understood as forming different algebras with some differences in their axioms. A theoretical understanding here will help to understand the peculiarities of the number systems used by computers with their limited precision (this will increase in importance as the new IEEE standard for real operations is used more often).

9.3 Topology
Many branches of engineering science deal with topological structures (electrical and other water networks, critical path, etc.). The Computer Aided Designing systems now available to manipulation of geometric or graphical structures rely (open or covered) on topological principles.

9.4 Theory of formal languages and automatons
Programming languages and, in general, all user interfaces to computer programs can be considered as formal languages. Theory can supply us with a few basic tools to describe a formal language (production rules, generally in the form of syntax diagrams), and to classify formal languages according to criteria of prime importance when the program to interprete the language is written. Similarily, some fundamental concepts from the theory of finite state automatons, the background for most theoretical studies in computer science, can furnish better models to illustrate computers' operation.

We do not propose that all these subjects from the more abstract parts of modern mathematics should be included. We do propose that we should critically examine the role of mathematics in the engineering curriculum. When we reconsider the place of computers in the curriculum, it is obviously appropriate to reconsider mathematics as well. We would advocate a much closer relationship between the two, and assume that computers can help students solve complex mathematical problems and, therefore, foster learning.

10. Conclusions

This paper presents a philosophy of computer education for engineering or science departments. It should be clearly noted that we do not advocate a single software, hardware or teaching technique, nor an isolated change in the curriculum. Such simple solutions are never adequate to coping with such complex a problem as integrating computing into education. We hope that we have convinced the reader that the measures we have taken are promising, namely:

- integrating computer usage in many courses so that students take at least one course with a strong computer usage component each semester
- expanding the introductory course from a 'programming language course' to a general course about computers and the principles of computer science
- teaching a programming language and a program development method which can be used for solving large technical problems
- selecting examples while teaching the programming languages that are related to engineering and science and form useful building blocks for the student for later use in problem solving.
- use interactive systems and computer graphics to make complex abstract topics visible (which is only possible on the most modern personal computers with the 'visible human interface' [9] and a consistent and simple command language).

Experience so far shows enthusiastic response from students and faculty in the course, but longterm success or failure of the experiment will depend on the faculty of following courses enhancing their teaching with computer related assignments.

References:

[1] Atkinson, L.U.; Harley, P.J.; An Introduction to Numerical Methods with Pascal, International Computer Science Series, Addison-Wesley Publishing Co., London, 1983

[2] Bowles, K.L.; Franklin, S.D.; Volper, D.J.; Problem Solving Using UCSD Pascal, Springer Verlag, New York, 1984 (2nd Edition)

[3] Brown, Marc H.; Sedgewick, R.; Technical Report CS-83-28, Brown University, R.I., 1983

[4] Chapra, S.C., Canale, R.P., Numerical Methods for Engineers, McGraw Hill, 1985

[5] Dahl, O.J.; Dijkstra, E.W.; Hoare, C.A.; Structured Programming, Academic Press, New York, 1972

[6] Dijkstra, E.W.; A Discipline of Programming, Prentice-Hall, Englewood Cliffs, N.J., 1976

[7] Feurzeig, W.; Lukes, G.; Lukes, J.O.; The LOGO Language, 1973

[8] Garlan, D.B.; Miller, P.L.; GNOME: An Introductory Programming Environment Based on a Family of Structure Editors, ACM Software Engineering Notes, Vol. 9, No. 3, May 1984

[9] Lu, Cary; Mac: The Apple Macintosh Book, Bellevue, Washington, Microsoft Press, 1984

[10] Papert, S.; Mindstorms: Children, Computers and Powerful Ideas, Basic Books, New York, 1980

[11] Parnas, D.L.; A Technique for Software Module Specification with Examples, Communications of ACM, Vol. 15, No. 5, May 1972, p. 330

[12] Parnas, D.L.; On the criteria to be used in Decomposing Systems into Modules, Communications of ACM, Vol. 15, No. 12, Dec. 1972, p. 1053

[13] Smith, D.C.; et al; Designing the Star User Interface, Byte Magazine, Vol. 7, No. 4, April 1982

[14] Syslo, M.; Deo, N.; Kowalik, J.; Discrete Optimization Algorithms with Pascal Programs, Prentice Hall, Englewood Cliffs, N.J., 1983

[15] Wirth, N.; Systematic Programming: An Introduction, Prentice Hall, 1973