DOCUMENT RESUME

ED 398 878 IR 018 058

AUTHOR Allan, V. H.; Kolesar, M. V.
TITLE Teaching Computer Science: A Problem Solving Approach that Works.
SPONS AGENCY National Science Foundation, Arlington, VA.
PUB DATE 96
CONTRACT DUE-9254186
NOTE 9p.; In: Call of the North, NECC '96. Proceedings of the Annual National Educational Computing Conference (17th, Minneapolis, Minnesota, June 11-13, 1996); see IR 018 057.
PUB TYPE Reports - Descriptive (141) -- Speeches/Conference Papers (150)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Academic Achievement; Algorithms; *Computer Science Education; *Course Content; Higher Education; *Instructional Effectiveness; *Introductory Courses; Problem Solving; Programming; Spreadsheets; Teaching Methods
IDENTIFIERS Internal Representation; *Preparatory Studies; Propositional Logic; Utah State University

ABSTRACT

The typical introductory programming course is not an appropriate first computer science course for many students. Initial experiences with programming are often frustrating, resulting in a low rate of successful completion, and focus on syntax rather than providing a representative picture of computer science as a discipline. The paper discusses the design of a preparatory course (CS0) to be taken before the introductory computer science course (CS1) at Utah State University. In the course, students gain mathematical and problem-solving skills while becoming familiar with the computer as a tool and learning how applications can save them time and work. Students are taught basic computer science concepts: data types, internal representation, operators, propositional logic, algorithms, control structures, programs, sub-programs, and recursion. Instruction is through lectures, readings, pencil-and-paper exercises, spreadsheet labs, reading and modifying code, and programming in the language ML, which identifies data type as the data are entered. The paper compares the performance of students in CS1 who took the preparatory course with those who took a course in BASIC prior to CS1. Fifty-three percent of the CS0 students who took CS1 achieved A grades in CS1, while only 26% of students who took BASIC before CS1 achieved an A grade in CS1. Results suggest that the preparatory course, with its skill-based approach to computer science, is a better preparation for the traditional first programming experience than a prior programming class. (Contains 13 references.) (SWC)

ED 398 878

Paper

Teaching Computer Science: A Problem Solving Approach that Works

V.H. Allan and M.V. Kolesar
Department of Computer Science, Utah State University
Logan, UT 84322-4205
801.797.2022/801.797.2421
allanv@cc.usu.edu
mvk@cc.usu.edu

Key Words: **pre-programming, problem solving, teaching, computer science**

Abstract

The typical programming course is not an appropriate first course for many students. Results indicate that our skill-based approach to computer science is a better preparation for the traditional first programming experience than even prior programming. (This work was partially supported by the National Science Foundation under grant DUE-9254186.)

1. Introduction

As evidenced by typical texts used in the CS1 first course in computer science, the traditional method of introducing computer science concepts is through programming. We believe that several factors indicate another approach is justified if we are to meet the needs of a diverse student population. At many institutions, the range of students, in terms of preparation, abilities, interests and motivation, is wide. Figure 1 shows a low rate of successful course completion (final grade of C- or better) in the first programming class. Note, over half of the majors in the College of Business and "other" colleges were unsuccessful. In addition, there is poor retention of students in the successor courses; for example, one academic session saw 204 students enroll in the second quarter of the CS1 sequence and only 107 complete.

Our contention is that a programming course used as the first exposure to computer science not only discourages many who are overwhelmed by what is required, but that it gives a distorted picture of the profession. For many students in CS1, it appears that rather than learning the basic concepts of the field, their energies are devoted to learning syntax. Rather than learning real problem solving skills, they resort to trial-and-error. Rather than "getting the big picture" of computer science, they narrow their focus to "getting this program to run."

Our data indicate that students with minimal prior computer experience face an almost insurmountable challenge in CS1. When the success of students was compared by experience level (Figure 2), only 22% of those with minimal previous experience were found to be successful in completing the course. One surprising fact is that any kind of computer experience (word processing, game playing, spreadsheets) is helpful;

over 50% of students with merely applications experience were successful in the first programming course. Since many of these activities have no established relationship to programming, one may conclude that the learning curve of computer familiarity is too steep to be scaled quickly by many students.

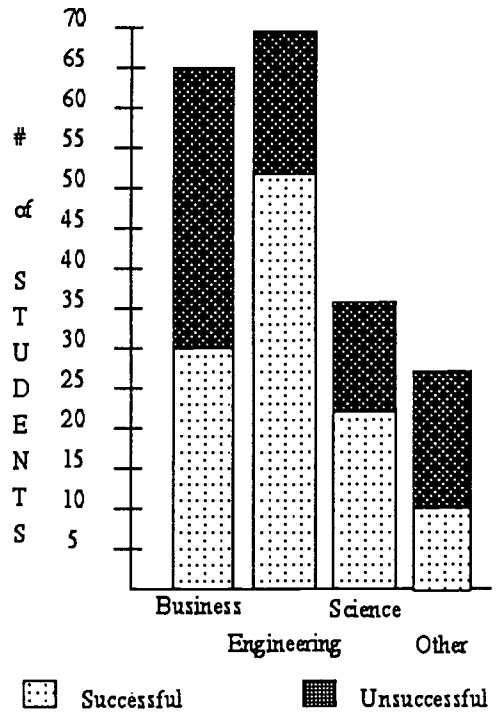

Figure 1. Performance in CS1 by college

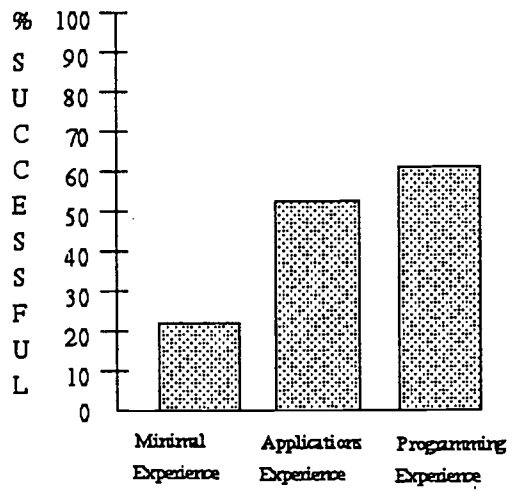

Figure 2: Relationship of prior experience to success in CS1

Students in a typical CS1 course must learn about the computer system, the programming environment, algorithmic problem-solving, top-down design in addition to the basic concepts like booleans, control structures, and mathematical representation, to

name a few. Finally, they must appropriately apply all this knowledge, and more, to the solution of a problem, while at the same time attempting to meet the requirement of syntactical correctness. Obviously, there are many students who are capable of this feat. But there are also many who are not capable or are not willing to expend the vast amount of effort needed. The latter make up our target population.

Many students in the programming sequence, even those who do well, report they find the sequence overly time-consuming and difficult. The ratio of time spent to concept learned is high; there is a great deal of frustration involved. If we consider the knowledge gained per hour of effort, programming without proper preparation ranks very low.

An additional factor indicating the need for another approach to introductory computer science is the low number of women seen in CS1. Of 322 students completing the first programming course less than 15% of the students were women (Statistics were taken from the 1993–1994 calendar year.). Historically, the percentage of women in computer science has been low, especially considering that mathematics experiences little gender difference [Fre90]. At Utah State University where 47% of the student body are women, 49% of 130 math majors are women. In math education, 71% are women, whereas in the traditional math major 40% are women. In contrast, of 243 computer science majors at USU, only 12% are women. At the graduate level, the percentage of women increases slightly to 14% of the 142 graduate students.

2. A Preparatory Course

We have developed a preparatory course, CS0, to be taken before the CS1 course. The course includes a laboratory experience, in keeping with recommendations found in the report of the NSF Workshop on Undergraduate Computer Science Education [NSF89]. Our intent was to develop a course that would stimulate the students to understand the issues before being asked to adopt a particular set of rules, to cause them to ask the question before being given the answer. Our goals were to furnish a firm foundation in the mathematical and problem-solving skills, teach some of the major computer science concepts needed for success in CS1, and give the hands-on experience that seems important for such success.

Class time is divided between lecture and work in small groups. Student interaction is highly encouraged. Instead of relying solely on the traditional lecture method where material is laid out for the students, we incorporate the Socratic method, where concepts are developed by questioning the students and building on their answers. This approach is quite successful in the humanities [Bat90] and has been used successfully for teaching mathematics [Pol71].

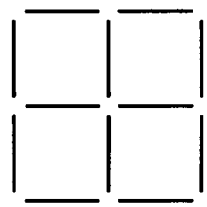

Figure 3: Move three toothpicks to create three boxes. All toothpicks must be used.

3. Course Strategies

The way we have juxtaposed ideas and slanted the coverage of certain topics differs from the norm and may serve as a launch pad for others' ideas.

3.1 Problem Solving and Problem Solving Strategies

Some students come to the first programming course without adequate problem-solving skills or with a poor understanding of the basic mathematical and logical tools needed for problem-solving. This situation has been recognized at all levels of education [Boa86]. The problem-solving experience of many students is confined to rote application of rules or use of standard equations.

In addressing the question of fostering problem-solving skills, it is our intent to motivate student interest by using a "recreational math" approach which has been used successfully in formal mathematics [Rub86, AC80]; challenge students with "brain teasers" and then give them the analytical tools to make solution easier. We attempted to implement the approach of "example, imitation and practice," which, according to Polya, is the only way to learn problem-solving [Pol62]. However, the problems are such that they have no specific solution pattern to follow. Rather than memorize, "When a problem has this characteristic, the solution looks like this," students practice the development of skills that apply to a wide range of problems.

We compiled a list of the skills used by good problem solvers [LGK89, Lev88, Man89, Pol62, Rub86, Suy80], then selected and designed activities and problems to foster those skills. This list includes:

1. They use self talk. They ask themselves questions, think aloud, and brainstorm.
2. They begin with what they understand and use all information.
3. They have acquired the knowledge of necessary subject matter.
4. They refer to the problem statement frequently.
5. They draw on other information and knowledge.
6. They employ lengthy sequential analysis when a question is initially unclear.
7. They create mental pictures and use physical aids to thinking.
8. They relate problems to familiar or concrete experiences.
9. They carefully proceed through a series of steps to reach a conclusion.

One particularly effective technique, self-talk, is stressed as students are encouraged to think out loud, ask themselves questions, recap what they have done and why. Many students have never thought about how they solve a problem. Thus, requiring them to vocalize the process is useful.

An exercise in problem solving uses the pattern of toothpicks and the problem statement shown in Figure 3. As students ask themselves a series of questions, the solution emerges. Do all the squares have to be the same size? (Probably. Only three can be moved, so there is little opportunity to construct different sized squares.) Can I have some lines hanging off? (No. All lines must be used.) Can I put toothpicks on top of each other? (Maybe, but that approach would only move two toothpicks.) Since each square takes four lines and there are twelve lines total, it appears that there are no common sides. (Yes, this is the beginning of the solution.) Throughout the course, as the class worked on problem-solving activities, the instructors encourage presentation of different approaches. Students are invited to journal their homework problem-solving activities.

A major problem-solving assignment involves the use of a spreadsheet. We ask students to analyze a set of demographic data collected on class members the first week of class. After learning the basics of using the spreadsheet package, students meet in study groups to discuss data analysis and to devise strategies for using the data. Students receive a copy of the assignment along with a study guide that includes discussion questions that attempt to stimulate critical thinking [KA95].

3.2 Computer Science Principles

Our approach to introducing basic concepts needed for computer science such as data types, internal representation, operators, propositional logic, algorithms, control structures, programs and subprograms and recursion is to use multiple ways of presenting the concepts. To present these concepts, we use various combinations of lecture, readings, pencil-and-paper exercises (individual and in small groups), spreadsheet labs, reading and modifying code (JoJo Dancer) and programming in ML. This multifaceted approach enables us to reinforce the perception that it is the concept itself that is important, not the syntax. Some specific examples follow.

The similarity between the spreadsheet cell and a variable is easily seen and provides a simple way to introduce variables. Constants are illustrated using the "cell protection" feature of the spreadsheet. Since a spreadsheet cell can contain an alphanumeric label, a numeric value or a formula, the concept of data types is motivated. Students see the need for data types before being asked to supply them in a program. The necessity of explicitly indicating strings can be shown with label entries consisting of digits. For example, the entry of a phone number, 555-5555, displays -5000, unless it is specified as an alphanumeric. By changing the display format of a numeric entry, the student comes to consider how a value entered as 24.329 can be displayed as an integer, a fixed-point value, or in scientific notation.

ML identifies data types as the data are entered, giving an immediate reinforcement of the data type concept, and also permits the definition of enumerated types and lists. The latter feature allows the instructors to emphasize the difference between the data type and an instance of that data type. After having manipulated integers, reals and strings, the student experiments with the enumerated type, gender, with values defined as male and female. Students are asked to explain the difference between male and "male."

While arithmetic operators are natural to students, the concept of relational and logical operators that produce boolean results is difficult for many students. Spreadsheets are an excellent way to introduce booleans and some basic logic at the same time. Seeing the evaluation of a relational expression displayed as 0 (false) or 1 (true) is a concrete, meaningful illustration for most students. The same is true in ML which displays the value true or false along with the type bool of an expression. Construction of truth tables in the spreadsheet is quick and allows evaluation and investigation of complex expressions. One useful example is the demonstration of DeMorgan's law as students are asked to demonstrate that not (a<b or c=0) does not have the same truth values as (a, b, or c 6= 0).

The built-in spreadsheet functions like SUM and AVERAGE serve to introduce the concept of list operations as well as to facilitate the presentation of subprograms and parameters. In this context, it makes perfect sense to students that when they ask for the SUM, they must specify what is to be summed. ML, as a functional language, is especially well-suited for an exposition of parameters. In addition, ML has a built-in list data type and has list operations defined, (e.g. hd and tl).

Common events are the keys which open the door to the study of algorithms. An in-class activity has students design an algorithm for roasting a marshmallow. Students are exposed to the concepts of conditionals and loops controlled by compound conditions without being burdened with having to understand the problem statement. Improper design of loops leads to "unhappy campers" who end up eating charred remains! Having a student (or, even better, the instructor) follow a student-designed algorithm for tying a shoelace leads to some humorous contortions. It is not until later that more formal algorithms designed for machine-solution are introduced.

An introduction to spreadsheet macros serves as a powerful, hands-on illustration of the value of creating reusable procedures. The idea is further enhanced in ML where the

function big (find the larger of two numbers) is used to write the function biggest which finds the largest of 4 numbers. It receives its fullest expression in JoJo Dancer, where the entire program consists of function calls. The fact that the student never sees the code for these functions highlights procedural abstraction. In lab, students follow instructions for the construction of a subprogram using the pre-coded functions and are asked to develop an original subprogram in a homework assignment.

Control structures are covered in numerous ways. Use of the spreadsheet IF function and the if-then-else in ML allows students to examine the selection control structure. In the spreadsheet, the capability of copying formulae with relative cell references makes the demonstration of iteration very simple. For example, the Fibonacci numbers can easily be generated iteratively: in the first two rows of a column (A, for example), enter the first two Fibonacci numbers (both 1); in the third row, enter the formula to add the previous two numbers(A3 =+A1+A2); and, copy the formula into as many cells in the column as desired. An investigation of the formulae stored in various cells, e.g. A7=+A5+A6, points out the repetition of the same operation with different values. Construction of other series, such as the series approximation of ex, allows the student to "program" relatively complex iterative problems with little concern for syntax. Comparing the results obtained by "programming" the series for x= -.01, and that obtained from the spreadsheet's built-in function EXP(-.01) leads to a discussion of round-off error and the effects of different computational methods.

In addition, control structures are explored as students read and modify C code which manipulates a graphical dancer. The use of single statements, composite statements and nested loops is seen as students modify the number of kicks, splits, or head tilts which JoJo performs.

ML, with its list data type and operators, makes recursion easy. We have found that by introducing recursion right along with iteration, students are open to accepting it as just another approach to solving a problem and acquire a good grasp of the notion.

	#	Precursor GPA	CSI GPA	% A's CSI
All CSI	683	none	2.6	30
Basic	92	3.0	2.3	26
CSO	30	2.8	3.2	53

Figure 4: Benefits of CS0 as a Preparatory Course

4. Results

In order to judge the effectiveness of CS0 as a preparation for CS1, a variety of statistics were gathered. We compared the performance in CS1 of students who took CS0 prior to CS1 to that of students who took the BASIC course prior to CS1. Our conclusion is that, not only is CS0 a good preparation for CS1, it is better preparation than a programming course.

Data for students who took the first quarter of CS1 during the period from Winter 1994 through Spring 1995 were analyzed. Students who had previously taken either CS0 or BASIC were noted. Data were collected on 30 CS0-prepared students and 92 BASIC-prepared students.

Figure 4 shows the benefits of CS0. On a four point scale, the average grade for CS1 over the period indicated was 2.6. CS0 students had an average grade of 3.2 in CS1, more than a half grade higher! This is even more impressive when one considers that those who took a college BASIC class before CS1 had an average grade of 2. in CS1. Fifty-three percent of CS0 students who took CS1 got A grades in CS1. Only 26% of the students taking Basic before CS1 received an A grade in CS1. It is interesting that those with BASIC preparation have a lower GPA in CS1 than the average student. This may indicate that students who feel they need a precursor class before taking CS1 are generally weaker and do not perform as well as other students, even after having taken the preparatory course. If this is true, it is even more impressive that CS0 students, who arguably felt a need for preparation, went on to out-perform the average student in CS1.

Since students at any level will spend hours exploring concepts they perceive as entertaining, we tried to motivate learning through a "recreational" approach. Judging from student involvement, the "brain teaser" approach did work. The class was most enthusiastic and responsive during these problem-solving activities. In terms of student interest and involvement, the spreadsheet is another successful class activity. Students report that they enjoy the satisfaction of learning a tool which can be applied to the solution of a variety of useful problems.

5. Summary

It is our experience that a typical programming course is not an appropriate first course for many students. Of all the activities in which a computer scientist is involved, learning a new language is one of the most frustrating. Even someone who has programmed in many other languages and knows what to do to solve the problem, finds it is necessary to stumble through manuals, searching under several possible topics, to find out how to do it. Since initial experiences with programming are often frustrating, resulting in a low rate of successful completion, another approach is warranted. Let the students experience good user-interfaces before they are asked to design one. Show the students how application programs save them work before asking them to spend hours coding a solution they can easily compute with a calculator. Introduce students to basic concepts like data types and round-off error through experimentation with a spreadsheet. Help the students develop problem-solving skills before asking them to apply those skills in a programming environment.

The preparatory class which we have implemented has several advantages. Students gain problem solving skills at the same time they are becoming familiar with the computer as a tool. They receive a more representative picture of computer science as a discipline without becoming confused by the details of syntax. Student understanding of top-down design, data structures, recursion, process abstraction, and other concepts central to computer science are of benefit throughout their undergraduate experience. Results indicate that this skill-based approach to computer science is a better preparation for the traditional first programming experience than even prior programming.

References

[AC80] B. Averbach and O. Chein. "Mathematics Problem Solving through Recreational Mathematics," Volume 1 of *Mathematical Sciences*. W. H. Freeman, New York, 1980.

[Bat90] W.L. Bateman. "Open to Question: The Art of Teaching and Learning by Inquiry." Jossey-Bass Publisher, San Francisco, 1990.

[Boa86] National Science Board. "Report of NSB Task Committee on Undergraduate Science and Engineering Education." March 1986.

[Fre90] K.A. Frenkel. "Women & Computing." *Communications of the ACM*, 23(11):34–46, November 1990.

[KA95] M.V. Kolesar and V.H. Allan. "Teaching Computer Science Concepts and Problem Solving with a Spreadsheet." In *Proceedings* of SIGSCE '95, pages 10–13, March 1995.

[Lev88] Marvin Levine. "Effective Problem Solving." Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[LGK89] F.K. Lester, J. Garofalo, and D.L. Krol. "Self-Confidence, Interest, Beliefs, and Metacognition: Key Influences on Problem-Solving Behavior," Chapter 6. In D.B. McLeod and V.M. Adams, editors, Mathematical Problem Solving: A New Perspective, chapter 6, pages 75–88. Springer-Verlag, New York, 1989.

[Man89] G. Mandler. "Affect and Learning: Causes and Consequences of Emotional Interactions." In D.B. McLeod and V.M. Adams, editors, Mathematical Problem Solving: A New Perspective, chapter 1, pages 3–19. Springer-Verlag, New York, 1989.

[NSF89] NSF. "Report on the National Science Foundation Disciplinary Workshops on Undergraduate Education." April 1989.

[Pol62] G. Polya. "Mathematical Discovery: On Understanding, Learning, and Teaching Problem Solving," Volume 1. John Wiley and Sons, New York, 1962.

[Pol71] G. Polya. "How to Solve It; A New Aspect of Mathematical Method." Princeton University Press, Princeton, NJ, 1971.

[Rub86] M.F. Rubinstein. "Tools for Thinking and Problem Solving." Prentice-Hall,Englewood Cliffs, NJ, 1986.

[Suy80] M.N. Suydam. "Untangling Clues from Research on Problem Solving." In S. Krulik and R.E. Reys, editors, Problem Solving in School Mathematics, chapter 5, pages 34–50. National Council of Teachers of Mathematics, Inc., Reston, VA, 1980.

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement (OERI)
Educational Resources Information Center (ERIC)

NOTICE

REPRODUCTION BASIS

This document is covered by a signed "Reproduction Release (Blanket)" form (on file within the ERIC system), encompassing all or classes of documents from its source organization and, therefore, does not require a "Specific Document" Release form.

☐

This document is Federally-funded, or carries its own permission to reproduce, or is otherwise in the public domain and, therefore, may be reproduced by ERIC without a signed Reproduction Release form (either "Specific Document" or "Blanket").