# Toward coherence in learning to program



**Authors**
Bork, Alfred
Pomicter, Nancy
Peck, Melissa
et al.

**Publication Date**
1984-07-26

# TOWARD COHERENCE IN LEARNING TO PROGRAM

Alfred Bork
Nancy Pomicter
Melissa Peck
Shawn Veloso

Technical Report 231

Department of Information and Computer Science
University of California
Irvine, CA 92717
(714) 856-6945

July 26, 1984

We see in the school systems in the United States and elsewhere a tremendous current urge to teach programming at a variety of levels. Sometimes this is done simply because of a general belief that programming is ``good'' for the students. Sometimes it is done under the guise of increasing reasoning capabilities, or problem solving capabilities, or creative capabilities, or intelligence, although little empirical evidence exists for any of these. Sometimes it is done under the guise of ``computer literacy'', although many other things are often given under that rubric, with no consensus as to what should be included. Sometimes it is done for vocational reasons.

We do not discuss the question of whether programming should be taught or at what grade level it should be taught. All those are interesting issues, but they seem to be very difficult issues to decide. Rather our question is, if programming is taught, what should be the criteria that determine what happens.

We do not intend to restrict ourselves in any way whatsoever to the currently available languages, systems, or approaches. We would argue that most of these strategies are inadequate. Professionals in computer science have given little thought to how programming should be taught, if it is taught at all. Most of the ``advisors'' in the school situation are not computer scientists, and most computer scientists, except for complaining about the teaching of inferior languages such as BASIC, have had little to say about the situation.

Given this background it is not too surprising that the existing programming courses in elementary and secondary schools have very little coherence. Often a variety of languages are used, with no reason why any particular one should be used except that it exists. Very seldom are these courses based on good modern structured programming philosophy and style.

The situation, however, is changing. More consideration as to what should be happening is occurring. Thus the recent interesting development of Boxer at MIT represents an attempt to think through some of the problems associated with children learning to program and to produce a superior environment.

We have also been engaged during the last few months in a project of this kind. The purpose of this paper is to discuss some of the philosophical considerations which should be taken into account when developing material to teach programming in the school system, and to outline the direction we are following.

## CONSIDERATIONS IN LEARNING TO PROGRAM

The following considerations seem to us important in thinking about how programming should be introduced and taught, with particular emphasis on starting in early grades. As already mentioned we are bypassing the more difficult issue as to whether programming should be taught at all, and we are assuming the decision has already been made that programming is to be an integral part of the school curriculum relatively early in the child's history.

1) Rapid evolution of knowledge about computers. Any program for teaching about programming must take into account that computing is a very dynamic area, and that therefore hardware, languages, approaches to learning languages, programming style, and many other factors are in a very rapid state of evolution. We should be paying close attention to this dynamic development, so as to avoid teaching obsolete strategies to young children.

2) Good programming and problem-solving strategies. It is important that from the earliest days students should be taught according to sound stepwise refinement standards of programming and problem solving. The materials that are developed should consider this a primary issue.

3) Strongly motivational approaches. We want to encourage students to be interested in the process of programming and to see it as an exciting activity. Hence, whatever learning strategies are employed need to carefully consider the motivational issues. This means that they should not just be motivational for a few good students in the program, but rather for all students.

Motivational concerns cannot be too highly stressed. Many studies continually show that the most important variable in education is time on task, so increasing time on task is important.

4) Curriculum materials are necessary. Most American classes, most subject areas, from kindergarten through the university, depend on the existence of textbooks or other

curriculum material. While romantically we can think of students and teachers proceeding entirely on their own, and this does happen in a few situations, for the vast bulk of our programs we need curriculum material.

In a new area such as programming, teachers will be particularly dependent on the existence of good curriculum material. Furthermore having such material will make it possible for the programming course to be used at home or in public libraries.

One thing that has been missing in much of the previous development has been a focus on the learning material itself. Indeed many past developments, such as Logo and Karel the Robot have developed the facility or language, the tool, and then only in a secondary way given thought to how that tool was to be used in classes. We can see some of the results of the lack of curriculum material if we look at the wide discrepancy between the potentials of Logo, and what is actually happening with Logo in most public schools.

It seems critical to us that the question of usage must receive precedence at the very beginning of a project. We emphasize that the problems with the current teaching of programming in schools has to do with the inadequacy of current curriculum material. Hence we recommend that a project which looks new in this area must begin with consideration of how the students are to learn the material in all possible environments in schools, homes, and elsewhere.

5) Coherent program. In most subjects in the schools there is some continuity from grade to grade. If one looks, for example, at arithematic from kindergarten through the twelfth grade, there is a continuous progress, a curriculum that has some plan and development associated with it. Teaching programming does not have such a development at present. Indeed, it often seems as if each teacher is beginning over, ignoring what might have happened before. A continuous program beginning with young children is as important here as in other academic areas.

6) Visual programming environments. Perhaps the main lesson to be learned from the use of Logo, Karel the Robot, and the early Computer Power material which includes quilting and cartooning is that visual environments are often useful in beginning programming. The young programmer can see immediately, with interesting pictorial output, whether the program works or not.

The environments do not have to be as sparse as Logo's, where one starts with a turtle in the middle of a clear screen. It is possible to have a very rich background as a starting point which may have some advantages with young kids. Attractive backgrounds may also have motivational attributes.

It is not necessary to have a single type of background. There may be a variety of different visual environments to work in, and these may change in different parts of the instructional material.

7) Integrated computer environment. The student who is learning to program will also be using the computer for other important purposes, such as word processing. Even in learning to program, more than the programming language itself must be involved. Editors and file-handling capabilities may be needed, although they may be concealed from the student.

Many of the editors in use at the present time in programming environments for young people are quite inadequate, with not very much thought given to them. Peculiar control characters of various kinds, for example, are needed. The notion of integrated products has become popular in noneducational application areas. Thus we have spreadsheets, word processors, and data-based systems combined with graphics within single products. This trend may be especially useful in education.

Particular thought needs to be given to how word processing can be integrated into the same environment in which programming occurs. It seems certain that word processing will be used more frequently in the school situation. Therefore, the student should not consider word processing to be entirely separate from programming activities. Some consideration is already being given to the issue of trying to establish a uniform way of viewing the computer, for example, in such developments as that of Boxer.

8) Use of modern computing tools. A variety of modern tools, used in programming environments at the advanced level, are possibly useful with young people. For example, intelligent

editors, editors that know something about the structure of the programming language to eliminate the entering of incorrect syntactical structures, remove some of the burden of the syntax from the student. Graphical exhibits of how programs behave, showing changes in variables, are also useful. Sophisticated systems of this kind have been developed for intorductory programming, both in films (by Ron Paecker, at the University of Toronto) and in the materials developed at Brown University for the Introductory Programming Class.

A variety of other kinds of programming aids, such as interpreters, style analyzers and similar material, should be very useful also. The general consensus is that we should consider, in developing any new programming facilities for the schools, the full range of tools we already have available in computer science for assisting with programming.

9) Do not emphasize grammatical details. One of the major problems with much of the teaching of programming is the enormous emphasis on grammatical details, at the expense of programming style, problem solving, and other aspects of the computer.

10) One language or several? Many of the curriculums currently in use in the schools seem to be based on the notion that students need to see all kinds of languages. Indeed, in some cases they seem to include every language available on whatever machine the school has, under the assumption that somehow this is better for the student.

We argue that the multiple language situation is unwise, unless the languages are very different, and therefore show very different aspects of how the computer works. For example, one could conceive of a programming environment which introduces students to both Ada and Prolog, because they are very different languages and demonstrate very different programming techniques.

11) Unsuccessful current situations. Looking at the actual teaching of programming within classes at the present time, it seems to us that the situation is often a disaster. The problem is that the teachers themselves have never programmed, and so have little understanding of what programming is all about. They think that a 10 or 15-line, linear, non-structured program is characteristic of programming and they teach accordingly.

12) Marketing issues. The development of instructional material for learning programming must also take into account how that material is to be marketed. Marketing in a new area, such as the use of computers in education, is generally fraught with problems. Marketing needs to be studied just as carefully as any other aspect of computer-based learning. We already have some disasters. It seems particularly ironic that one of the better pieces of curriculum material for teaching programming in the schools, the Computer Power material from McGraw-Hill, has been so poorly marketed that hardly any schools know of its existence. We frankly do not know the reasons for this.

13) Additional research. We very much need some careful research in the teaching of programming. These issues can only be

settled by careful research done on a larger scale than anything currently available.

LEARNING INTRODUCTORY PROGRAMMING BASED ON THESE CONSIDERATIONS

The topics just presented give a clear view as to many of the features that one would expect introductory sessions in programming, particularly for primary school, to have. We are developing a learning environment based on these considerations. While this environment is still in its early stages it is possible to give some description of its operation. Further details should be available for the conference. We should be able to run on-line demonstrations then.

1) The language and associated facilities. Two sets of material are considered. First there is the language and programming environment in which students program. It should ``grow'' with the user; it should, at each stage, understand the user's capabilities and knowledge of the language; and it should proceed accordingly. Particularly it should know what learning modules (to be described) this student has been through, and what progress has been made in each module. As indicated the learning environment will ultimately include intelligent editors which know something of the syntax of the capabilities provided.

Our plan is to base this language and facility on the syntax of Ada. Initial modules contain simplified constructs designed for beginners. Later modules progress to more powerful Ada constructs. A clear progression is possible throughout the student's career. (We are considering developing a parallel set of material involving a language such as Prolog at a later date.)

The language capability has many of the desirable features of Logo. We assume initially a simple visual environment to be described in connection with the learning material. We also assume an interpretative capability and the same type of automatic storage of student-developed procedures that is available in Logo. The facilities avoid control characters. It presents a highly friendly user interface even when using the language with no learning module.

2) Learning modules currently planned. The following learning modules are in various stages. Several of them have been designed, and some coding has taken place. Some of the latter ones have only been outlined. A further report on progress will be made at the conference.

In the first module we introduce and allow the user to play with two commands—MOVE and TURN. Neither one of the commands contain any arguments. Thus they are simpler than Logo commands. They do not, in this module, draw; rather they move a creature (of interesting visual perspective) around the screen. The two commands are introduced by simply letting the students play with them, with the computer watching over their shoulder to see how they are used.

The major difference in this introductory module from most introductions to Logo is that the material is self-contained, and that the creature moves around in interesting visual environments containing various objects. The creature can bump into the objects, and this is reported. Graphic artists are working with us in designing various environments.

A second module allows the user to use the Move and Turn commands with a specific task. The computer ``watches'' to see if the student can carry out the tasks, and to determine whether additional help is needed. This module serves as an alternate starting module since it is capable of providing all the instruction needed about the commands if necessary. A variety of tasks are provided in this module and assigned to students randomly. Different visual environments are involved in these tasks to increase student interest.

The next module is concerned with the fundamental ideas of problem solving within environments already introduced. The notion stressed will be breaking a sizable task into smaller tasks. The tasks involve moving the creature around a rich environment. Emphasis is NOT on writing the procedures necessary, but on defining the sub-tasks. Thus little in the way of actual coding or grammatical details associated with the language appear in this module. We believe that it is critical that students understand these issues in a language-independent form early. If students do not begin with a top-down point of view they may have difficulties mastering it later. Development of such tactics cannot be left to chance or to the possible motivations of an excellent teacher. Rather we want to make certain everyone, even those working at home alone, develop structured problem-solving strategies at an early stage.

In the next module, which immediately follows the one just described, the user constructs a main program to implement the task given in the previous module. Procedures in the main program correspond directly to the sub-tasks defined by the student. An intelligent editor will help with many of the details as the user is brought to view the main program as the mechanism for breaking up a large problem into smaller problems. Again all of the problems involve the visual environments and moving the creature around in those environments.

Subsequent modules introduce important programming concepts. We introduce loops using the most primitive Ada looping structure. The advantage of this structure is that it shows the exit explicitly. Issues as to where the testing occurs are not raised to confuse the early programmer. When we present a grammatical concept we present only enough of what is needed at a particular stage in the learning process, rather than trying to get people to understand ``everything'' about the concept at that stage. IF-THEN-ELSE statements are also approached in the same way and are used initially in connection with bumping into the objects in the environment.

These directions are only an early attempt. We expect to be refining some of these as we use the material with students. We want to develop a programming environment which will work with and without teachers, in schools and in homes, and in public places such as libraries.