Logo Programming and Problem Solving

Roy D. Pea

Technical Report No. 12

LOGO PROGRAMMING AND PROBLEM SOLVING*,**

Roy D. Pea

Center for Children and Technology
Bank Street College of Education

In the world of educational computing, programming is a major activity, occupying several million precollege students a year in this country alone. As yet very little is known about what kinds of cognitive activities computer programming requires and whether, in the classroom contexts that are representative of microcomputer use in schools today, children are capable of making substantial progress in learning to program. In the cyclical program-development process of problem understanding, program design and planning, programming code composition, debugging, and comprehension, what gains do children make on the many developmental fronts represented in the complex of mental activities required by programming? Do conceptual limitations impede their understanding of any of the central programming concepts, such as flow of control structures, variables, proceduarality, and the like? We have begun to address aspects of these questions in our developmental research on children learning to do Logo programming.

I would like to make five points which will be explicated in the remainder of this paper:

1. Systematic developmental research documenting what children are learning as they learn to program is necessary, rather than existing anecdotes. Our studies focus on Logo because it is a programming environment that is exciting to many educators, it has great potential for introducing children to many of the central concepts involved in programming and problem solving, and because grand

---

*Paper presented at symposium of the American Educational Research Association, "Chameleon in the Classroom: Developing Roles for Computers," Montreal, Canada, April 1983.

**The research reported in this paper was funded by the Spencer Foundation.

claims have been made for how it promotes learning to program, as well as metacognitive skills such as planning and strategic problem solving.

2. Logo is cognitively complex beyond its early steps, and quite difficult to learn without instructional guidance, even if students are intellectually engaged with that learning. While the semantics and syntax of Logo are readily learned, the pragmatics--how to arrange lines of legal programming code to achieve specific ends--is a great challenge.

3. The pedagogical fantasy (e.g., Byte, August 1982; Papert, 1980)--that Logo can serve as a stand-alone center in classrooms for learning programming and thinking skills--does not work. Teacher training will be necessary for programming skills to develop very far, and problem-solving skills may need to be taught directly rather than assumed to emerge spontaneously from learning Logo.

4. After a year's experience of programming in Logo, following the discovery-learning pedagogy advocated for Logo, two classes of 25 children (8- to 9-year-olds, 11- to 12-year-olds), each with six computers, did not display greater planning skills than a matched group who did not do Logo programming.

5. We need to develop an instructional science for programming if that is what we wish children to learn, but we also need to re-think, in ways suggested by Midian Kurland, the educational goals that programming is meant to fulfill.

The great excitement in education about children's learning to program with microcomputers is easy to understand. But it is of particular interest to me as a developmental psychologist that such excitement has had less to do with the practical value of learning how to write programs for specific applications than with the belief that, through learning to program, children will develop powerful cognitive skills such as planning abilities, problem-solving heuristics, and reflectiveness on the revisionary character of problem solving itself (Pea & Kurland, 1983). This idea--that programming will provide exercise for the highest mental faculties, and that the cognitive development thus assured for programming will generalize or transfer to other content areas in the child's life--is a great hope. Many elegant analyses offer reasons for this hope, although there is an important sense in which the arguments ring like the overzealous prescriptions for studying Latin in Victorian times.

Programming is viewed by many of its devotees as a "Wheaties of the Mind," a panacea for the ambiguities of everyday cognition. It is alleged that in the demands which programming activities make on the mind--of precision (in requiring a specific sequence of instructions for controlling the operations of the computer); of problem decomposition (into component subproblems); and of debugging (systematic efforts to eliminate discrepancies between the intended outcomes of the program and those brought about through the current version of the program)--programming renders salient the general utility of such cognitive activities in problem-solving efforts, and that such generalizations will be made spontaneously by the programmer to problem spheres above and beyond the microcomputer environment (Feurzig et al., 1981; Minsky, 1970; Papert, 1980). To place these claims in a larger context, we may note their similarity to claims about how literacy, or mathematics, or logic serve as "cognitive amplifiers," enabling the users of such technologies to transcend the limitations of their previously available tools of thought (Bruner, 1966; Cole & Griffin, 1980; Goody, 1977; Olson, 1976).

What has been done to evaluate the empirical validity of these important claims? While Papert and colleagues undertook extensive studies of children doing Logo programming in the Brookline school system, their reports of this work were principally qualitative in nature, citing and discussing some of the programs that were created by the children, the global differences in programming style that seemed to be intuitively distinguishable (Watt, 1979), and dramatic case studies of great programming progress made by children who had learning difficulties (e.g., Weir, 1981).

Though interesting, these reports do not directly address the widely touted claims for the development of thinking skills that transcend the programming context, for which case-study methods are inappropriate. We thus undertook a series of investigations in order to provide systematic data on children learning to program and the alleged cognitive outcomes of such programming, such as developments in planning skill. Methodologies for addressing these questions were developed, and summaries of some key research findings to date are presented below.

Research

I will briefly review three of our studies. Detailed technical reports will be available in the near future. The first was a study of the level of programming expertise that children had developed by year's end; the second consisted of systematic probes of the depth of understanding of programming concepts such as "recursion" in studies with

individual children; and the last asked whether children doing programming developed planning skills that they then spontaneously transferred beyond programming.

In one study, we presented children with a 3-part written assessment of programming. The three parts, each taking 45 minutes, were: (1) Logo command understanding, where children were asked to fill in, with graphics or words, what would happen on the screen when specific commands were typed and entered into the computer; (2) writing Logo programs to draw shapes, with constraints as to what programming constructs (e.g., tail recursion, variable) were to be used; and (3) finding the errors or bugs in prewritten programs intended to achieve a pictorially specified goal. For command understanding, we found that, although the number of hours spent programming by the older (25 hours) and younger (29 hours) groups were not significantly different, the older group understood significantly more commands than the younger children. Boys spent more hours programming (34 vs. 22 for girls), and outscored the girls on nearly every class of programming commands. Performance on this command comprehension task was revealing: out of 100 possible points, the mean score for commands understood in terms of this measure was 34, with a huge standard deviation of 25, and only three out of the 50 children scored between 75 and 95. Roughly one-quarter of the children in each of the classes had not become very much involved in the classroom programming and did correspondingly poorly. In the case of writing different programs that would each draw a box of a certain size, we found that, while few children had difficulty writing a program consisting of a chain of direct commands (FD, RT) or a tail recursive procedure, many children could not write a version of such a program using a variable, or a version of the tail recursive program with a conditional test that would stop the drawing. In the area of debugging, many children were able to locate and eliminate "surface" errors of syntax, or missing variable values, but very few found procedural errors in which the order of lines in a program was mixed up.

The second study (Kurland & Pea, 1983) utilized a series of increasingly complex Logo programs that were designed to reveal the depth of understanding of recursion in a half dozen of the best programmers in the two Logo classrooms. The method we used--having children read through the programs line by line and make predictions as to what would happen when each line of programming code was executed--was extremely telling, and confirmed our classroom observations. Four prevailing tendencies are of central developmental significance. One was to treat the program as akin to natural language text, ambiguous in meaning and "ignorable" by the computer if

the child did not understand it. A second was the fact that some children did not understand conditional test statements in these programs even though they had written programs that contained them. This is a robust finding, as other studies with these children have shown; the children's programs often displayed production without comprehension, in that programming constructs such as variables, test statements, or even simple commands like "repeat" may have been used in one program, but not understood in another. Rote use of "chunks" from other children's programs or those of the teacher seems to be responsible for this rigidity of use. A third tendency was to violate the sequentiality of program execution, to assume that, without instructions to do so, the computer could "jump" some lines in the program to execute other lines. The fourth tendency, common to all the children, was to manifest a misguided mental model of how recursion works, one which is insurmountable without instruction since, for recursion, evidence for how control is passed in Logo (i.e., which line is to be executed next) is not discoverable in the surface structure of the language.

The third study was a longitudinal pre-post investigation of groups of children who were provided with extensive opportunity to program in the Logo language over a school year. These children were then compared to a matched group of nonprogramming students to see whether learning to program enhanced the development of planning skills. The task, administered before and after the year of programming, was a classroom chore-scheduling task that allowed children multiple opportunities to come up with the shortest plan they could construct for carrying out a series of chores. Our expectation was that better planners would take a more strategic approach to the task, revise or debug their plans more effectively, and engage in more executive and metacognitive decision making as they developed their plans (Hayes-Roth & Hayes-Roth, 1979; Pea, 1982). On a large number of measures--the efficiency of the plans, the quality of the revisions, and the types of decisions made during the planning process--we found no differences between the programming and nonprogramming groups at either age.

Why did we find no cognitive benefits on our task for those children who had been doing Logo programming for a year? Advocates of the cognitive benefits of programming might object that our treatment was not of sufficient duration for benefits to be manifested, or that benefits could be revealed in later years, but not so soon after the "treatment" provided by Logo.

However, we favor an interpretation more in keeping with two general findings in cognitive science during the past decade, and with addi-

tional observations of the children in our planning task while programming. The first finding is that transfer of problem-solving strategies between dissimilar problems, or problems of different content, is notoriously difficult to achieve even for adults (Gick & Holyoak, 1980; Hayes & Simon, 1977; Tuma & Reif, 1980). The second finding is that, even among computer science students in college who, by their junior year, have had several thousand hours of programming study (as contrasted with about 30 hours for our student groups), great conceptual difficulties in understanding how even brief programs are working persist (Soloway et al., 1982), which one would not predict if planning and problem-solving skills had achieved such extensive development by virtue of programming experience.

Our in-class observation had to do with whether children plan prior to programming. It has been an assumption of those expecting transfer of planning skills developed within programming to domains outside programming that, in fact, planning skills are at least developed in programming. But we found very little preplanning activity. Planning a program by specifying the high-level logic that a program would be written to implement was not a distinct component of the children's program development process. Much more common was on-line programming, in which children defined their goals, and found means to achieve them as they observed the products of their programs unfolding on the screen. Rather than constructing a plan, then implementing it as a program to achieve a well-defined goal, and afterwards running the implemented plan on the computer, children would evolve a goal while writing lines of Logo programming language, run their program, see if they liked the outcome, explore a new goal if they did not like the outcome by writing a new programming code, and so on. It might be objected that, although they engaged in little top-down planning, they did work a great deal on plan revisions by continually adapting their programs, revisions being central to planning activity (Pea, 1982). If this is so, we should have seen differences between the programming classes and the control classes in planning revisions during the noncomputer planning task, but we did not. And program debugging in the classrooms would have been very common. In most cases, children preferred to rewrite a program from scratch rather than to suffer through the attention to detail required in figuring out where a program was going awry. As one child put it when asked why she was typing in commands directly rather than writing a program: "It's easier to do it the hard way." Debugging requires precision and line-by-line program comprehension; in general, both were difficult for the school-aged children to attain after a full year. They certainly were not automatic consequences of exposure to Logo.

While we believe that, on the basis of these findings, it would be premature to discard programming or Logo from the set of microcomputer uses in schools, these studies do raise serious doubts about the sweeping claims made for the cognitive benefits of learning to program, particularly in Logo (see Byte, August 1982). We find that the entry level of Logo--getting the turtle to carry out mathematically interesting drawings through writing short programs consisting of one or two procedures--does not present conceptual problems for the school-aged child. Far from being problematic, one finds in most children just the mental engagement that advocates of Logo highlight. But the elegance and beauty of Logo that derives from its parent language, LISP, used in artificial intelligence, its procedurality which allows one to define new procedures and use them as building blocks in increasingly complex programs, its control structures that allow very brief recursive programs that can solve quite difficult problems, the use of conditional tests--all these features present deeply challenging conceptual problems on a turf our children did not opt to travel during their discovery learning. With thoughtful instruction, which will require developmental research for its design, we expect that Logo may provide a good window for the child into these important computational concepts. With accompanying instruction in thinking skills (see, for example, Chipman, Siegel & Glaser, 1983), perhaps using Logo or other programming languages as a vehicle for discussing heuristics and problem-solving methods, developments in planning skill may in fact be achieved. But we have deep doubts, based on a series of empirical studies over an 18-month period, that the Logo ideal is attainable with its discovery-learning pedagogy.

References

Bruner, J. S. On cognitive growth. In J. S. Bruner, R. R. Olver, P. M. Greenfield (EDs.), Studies in cognitive growth. New York: Wiley, 1966.

Chipman, S., Siegel, J., & Glaser, R. (Eds.), Thinking and learning skills: Current research and open questions. Hillsdale, NJ: Erlbaum, 1983. In press.

Cole, M., & Griffin, P. Cultural amplifiers reconsidered. In D. R. Olson (Ed.), The social foundations of language and thought: Essays in honor of Jerome S. Bruner. New York: Norton, 1980.

Feurzig, W., Horwitz, P., & Nickerson, R. S. Microcomputers in education (Report No. 4798). Prepared for: Department of Health, Education, and Welfare; National Institute of Education; and Ministry for the Development of Human Intelligence, Republic of Venezuela. Cambridge, MA: Bolt Beranek & Newman, October 1981.

Gick, M. L., & Holyoak, K. J. Analogical problem solving. Cognitive Psychology, 1980, 12, 306-355.

Goody, J. The domestication of the savage mind. New York: Cambridge University Press, 1977.

Hayes, J. R., & Simon, H. A. Psychological differences among problem isomorphs. In N. J. Castellan, Jr., D. B. Pisoni, & G. R. Potts (Eds.), Cognitive theory (Vol. 2). Hillsdale, NJ: Erlbaum, 1977.

Hayes-Roth, B., & Hayes-Roth, F. A cognitive model of planning. Cognitive Science, 1979, 3, 275-310.

Kurland, D. M., & Pea, R. D. Children's mental models of recursive Logo programs. Proceedings of the Fifth Annual Cognitive Science Society, Rochester, New York, 1983.

Minsky, M. Form and content in computer science. Communications of the ACM, 1970, 17, 197-215.

Olson, D. R. Culture, technology and intellect. In L. B. Resnick (Ed.), The nature of intelligence. Hillsdale, NJ: Erlbaum, 1976.

Papert, S. Mindstorms. New York: Basic Books, 1980.

Papert, S., Watt, D., diSessa, A., & Weir, S. Final report of the Brookline LOGO Project: An assessment and documentation of a children's computer laboratory. Cambridge, MA: MIT Division for Study and Research in Education, 1979.

Pea, R. D. What is planning development the development of? In D. Forbes & M. Greenberg (Eds.), New directions in child development: Children's planning strategies (Vol. 18). San Francisco, CA.: Jossey-Bass, 1982.

Pea, R. D., & Kurland, D. M. On the cognitive and educational benefits of teaching children programming: A critical look. New Ideas in Psychology, 1983, 1 (3). Elmsford, NY: Pergammon. In press.

Soloway, E., Ehrlich, K., Bonar, J., & Greenspan, J. What do novices know about programming? In B. Shneiderman & A. Badre (Eds.), Directions in human-computer interactions. Hillsdale, NJ: Ablex, 1982.

Tuma, D. T., & Reif, F. (Eds.). Problem solving and education: Issues in teaching and research. Hillsdale, NJ: Erlbaum, 1980.

Watt, D. A comparison of the problem solving styles of two students learning LOGO: A computer language for children. Proceedings of the National Educational Computing Conference, 1979, 255-260.

Weir, S. LOGO as an information prosthetic for the handicapped (Working Paper No. 9). Cambridge, MA: MIT Division for Studies and Research in Education, May 1981.