ED 249 930 IR 011 358

AUTHOR	Pea, Roy D.; Kurland, D. Midian
TITLE	Logo Programming and the Development of Planning Skills. Technical Report No. 16.
INSTITUTION	Bank Street Coll. of Education, New York, NY. Center for Children and Technology.
SPONS AGENCY	Spencer Foundation, Chicago, Ill.
PUB DATE	Mar 84
NOTE	57p.; For related document, see IR 011 339.
PUB TYPE	Reports - Research/Technical (143)
EDRS PRICE	MF01/PC03 Plus Postage.
DESCRIPTORS	Classroom Research; *Computer Assisted Instruction; *Critical Path Method; Elementary Education; Hypothesis Testing; Intermediate Grades; Longitudinal Studies; *Microcomputers; *Planning; *Programing; Skills; Tables (Data); *Transfer of Training; Use Studies
IDENTIFIERS	*LOGO Programing Language; Process Measures

ABSTRACT

Findings are presented from two separate year-long longitudinal studies of the development of planning skills among school aged children in relation to learning LOGO programming, and a theoretical context is provided for predictions of greater improvement by the programming groups. In the first year, experimental groups comprised students in each of two classrooms in a private school in Manhattan. One classroom included 25 8- and 9-year-old children; the other consisted of 25 11- and 12-year-old children. The control groups were made up of students in the same grade level classrooms in the same school. Both experimental groups were administered a classroom chore-scheduling planning task. Process and product measures of planning skill revealed no benefits for students doing LOGO programming. The second experiment took place one year later in the same school in the same two teachers' classrooms. The second study comprised 32 students in each of the age groups of the first study. This time a microcomputer version of the task was implemented in which students gave commands to a robot to carry out the chores, and similar assessments of planning performances were collected online. Again, learning to program did not differentiate experimental from control group performances. Further tests of the programming transfer hypothesis are proposed. Data tables and references are included. (Author/THC)

TECHNOLOGY

Logo Programming and the Development of Planning Skills

Roy D. Pea and D. Midian Kurland

Technical Report No. 16

March 1984

LOGO PROGRAMMING AND THE DEVELOPMENT OF PLANNING SKILLS*

Roy D. Pea and D. Midian Kurland

Introduction

A belief which has never gone completely out of style is that learning mentally rigorous disciplines, such as logic, geometry, or Latin, exercises the mind such that general thinking powers are enhanced. Implicit in this belief is the idea that rigorous thinking in one discipline may transfer, or be generalized to, rigorous thinking in other disciplines. It is well recognized that transfer, particularly of intelligent performance to novel task environments, is the hallmark of human cognition (e.g., Anderson, 1982; Brown, Bransford, Ferrara & Campione, 1983; Laboratory of Comparative Human Cognition, 1983). To date, no studies have adequately demonstrated this transfer of mental rigor, however defined. In fact, Thorndike (1924) said he had put this claim to rest in his studies of students' performances in other disciplines after they had learned Latin.

Today we find resurgent interest in and deep commitment to a version of this belief cast in the context of new information technologies available to schools. In particular, does learning to program a computer provide such experiences in higher mental functioning that the rigors of computer programming transfer to thinking and problem solving in other areas? Many are optimistic that it does. Nickerson (1982) has characterized the current wave of opinion as follows:

> Perhaps the basic reason for the belief that programming might be an effective vehicle for the acquisition of generally useful cognitive skills is the assumption that programming is

*We would like to thank the students and teachers of the Bank Street School for Children for cooperating in these studies. Invaluable assistance was provided by Sally MacKain, Moni Hamolsky, and Jeff Aron in coding videotapes, and Sally MacKain was our tireless experimenter for study two. Jan Hawkins made continuing contributions to the project. These research studies were part of a comprehensive program on the development of programming skills in classrooms supported by the Spencer Foundation, to whom we are grateful.

> prototypical of many cognitively demanding tasks. It is a creative endeavor requiring planning, precision in the use of language, the generation and testing of hypotheses, the ability to identify action sequences that will realize specified objectives, careful attention to detail, and a variety of other skills that seem to reflect what thinking is all about.

Our paper is a report on several studies that evaluate the empirical validity of these expectations for the general cognitive effects of learning to program. Like any empirical study, they require a finer grained focus than these overarching ideas will allow and, in this case, we have examined the development of planning skills in relation to learning to program in the Logo programming language (Abelson & diSessa, 1981; Papert, 1980). We will return to a discussion of the mental activities involved in programming and our choosing to study planning as a higher mental function after briefly sketching out the larger terrain in which these concepts have become embodied in educational practices.

It is important to set the broader context for these claims about the cognitive benefits of learning to program. The widespread availability of microcomputers has led to their rapidly increasing use in the nation's schools, and a vast number of these schools are using computers primarily for instruction in programming. At all grades, but particularly at the elementary and secondary levels, the two reasons most frequently cited by schools for placing such a heavy emphasis on programming are its presumed impact on thinking skills beyond programming activities (e.g., Coburn, Kelman, Roberts, Snyder, Watt & Weiner, 1982), and its contribution to the student's "computer literacy," since programming is thought to be a major aspect of computer use.

Cognitive Consequences of Learning to Program

The current expectations and frequent claims for the effects of learning to program on thinking are best exemplified in the writings of Papert and Feurzeig (e.g., Feurzeig, Papert, Bloom, Grant & Solomon, 1969; Feurzeig, Horwitz & Nickerson, 1981; Goldstein & Papert, 1977; Papert, 1972a, 1972b, 1980; Papert, Watt, DiSessa & Weir, 1979) concerning the Logo programming language (developed in the last fifteen years as a learning environment for children: see Byte, August 1982), although such claims are not unique to Logo (see Minsky, 1970).

Two key catalysts appear to have contributed to the belief that programming may spontaneously discipline thinking. The first is from artificial intelligence, where constructing programs that model the complexities of human cognition is viewed as a way of understanding that behavior. The contention is that in explicitly teaching the computer to do something, you learn more about your own thinking. By analogy (Papert, 1972a), programming students would learn about problem-solving processes by the necessarily explicit nature of programming, as they articulate assumptions and precisely specify steps to their problem-solving approach. "In teaching the computer how to think, children embark on an exploration about how they themselves think" (Papert, 1980, p. 19). The second influence is the widespread assimilation of constructivist epistemologies of learning, most familiar through Piaget's work. Papert (1972a, 1980) has been an outspoken advocate of the Piagetian account of knowledge acquisition through self-guided problem-solving experiences, and has extensively influenced conceptions of the benefits of learning to program through "learning without curriculum" in "a process that takes place without deliberate or organized teaching" (1980, p. 8; also pp. 27, 31). (It should be noted that Piaget never advocated the elimination of organized teaching in schools.)

Ross and Howe (1981) have summarized Feurzeig et al.'s (1969) four expected cognitive benefits of learning to program. In this early period, most programming outcomes were postulated for the development of mathematical thought:

> (1) that programming provides some justification for, and illustration of, formal mathematical rigour; (2) that programming encourages children to study mathematics through exploratory activity; (3) that programming gives key insight into certain mathematical concepts; and (4) that programming provides a context for problem solving, and a language with which the pupil may describe his own problem solving. (p. 143)

Papert (1972b) argued for claims (2) through (4) in noting that writing programs of Logo turtle geometry is a

> new piece of mathematics with the property that it allows clear discussion and simple models of heuristics [such as debugging] that are foggy and confusing for beginners when presented in the context of more traditional elementary mathematics.

He provides anecdotes of children "spontaneously discovering" phenomena such as the effects of varying numerical inputs to a procedure for drawing a spiral on the spiral's shape. He concludes that learning to make these "small discoveries" puts the child "closer to mathematics" than does faultlessly learning new math concepts.

It is important to distinguish between the expectation that programming can discipline thinking, and the prediction that programming will discipline thinking. We consider the possibility argument very plausible, especially if thinking skills are explicitly taught in the context of programming, but this experiment has never been carried out. The much stronger predictive claim, best known from Papert's (1980) arguments, is more commonly assumed--that programming will discipline thinking through the child's spontaneous explorations and reflection on programming activities. Although our can/will distinction may seem to split hairs, it is central for any empirical work on the question, since it will determine the character of the "treatment" for the experimental groups who are programming. Our experience has been that the pedagogy of discovery learning ("learning without curriculum") for Logo programming mapped out in Papert (1980) is taken quite seriously by teachers, and that little direct instruction of computational concepts, programming methods, or concepts of a theory of problem solving is offered when Logo programming is undertaken in school settings. We see it at its extreme when teachers talk excitedly about how Logo can be used in "stand-alone" centers within classrooms.

Within the last several years, and in the absence of experimental support for the first series of claims (see reviews in Pea & Kurland, 1984; Ross & Howe, 1981), much broader claims for the cognitive benefits of programming beyond mathematics have been made in a new generation of theoretical writings. In Mindstorms, Papert (1980) argues that generalizable cognitive benefits will emerge from taking "powerful ideas" inherent in programming (such as recursion and variables) in "mind-size bites." Feurzeig et al. (1981) provide the most extensive list of cognitive outcomes that may emerge from learning to program. They argue that

> the teaching of the set of concepts related to programming can be used to provide a natural foundation for the teaching of mathematics, and indeed for the notions and art of logical and rigorous thinking in general.

It is important to note that their emphasis on teaching distinguishes their discussion of programming benefits from Papert's recommendations for learning to program "spontaneously" without curriculum (1980). We review their claims because they are the clearest articulations to date of what general changes in higher mental functioning may be expected from learning to program. Seven fundamental changes in thought are described, which we summarize and explain here:

(1) rigorous thinking, precise expression, recognized need to make assumptions explicit (since specific algorithms must be written for programs to work);

(2) understanding of general concepts such as formal procedure, variable, function, and transformation (since these are used in programming);

(3) greater facility with the art of "heuristics," explicit approaches to problems useful for solving problems in <u>any</u> domain, such as planning, finding a related problem, solving the problem by decomposing it into parts, etc. (since "programming provides highly motivated models for the principle heuristic concepts");

(4) the general idea that "debugging" of errors is a "constructive and plannable activity" applicable to any kind of problem solving (since it is so integral to the interactive nature of the task of getting programs to run as intended);

(5) the general idea that one can invent small procedures as building blocks for gradually constructing solutions to large problems (since programs composed of procedures are encouraged in programming);

(6) generally enhanced "self-consciousness and literacy about the process of solving problems" (due to the practice of discussing the process of problem solving in programming by means of the language of programming concepts);

(7) enhanced recognition for domains beyond programming that there is rarely a single "best" way to do something, but different ways that have comparative costs and benefits with respect to specific goals.

<u>Planning</u>

For our current studies, we will be examining one aspect of claim (3)--the problem solving heuristic of planning. Why planning? Planning is a central aspect of intellectual functioning. Although, in everyday activities, problems that result from poor planning are readily attributable to other causes since feedback is delayed, Nickerson (1982) observes that "the immediacy of the feedback in programming contexts may help to make the causal link between adequate planning and effective performance more apparent" (p. 43). Does the effectiveness of planning become more apparent to a person learning to program? Does the development of planning skills for more general use as thinking tools become more likely when one learns to program?

Before moving on to planning and programming, we highlight a fundamental cognitive issue raised by asking whether programming promotes the development of any of these general cognitive skills. Is it reasonable to expect transfer (without overt encouragement) across knowledge domains, from programming to science research projects, math problem solving, or social affairs? It is notoriously difficult for people to spontaneously recognize the connections between "problem isomorphs"--problems of identical logical structure but different surface form (Gick & Holyoak, 1980; Hayes & Simon, 1977; Simon & Hayes, 1976)--and to apply problem-solving strategies learned in one context to new problem forms. When Scribner and Cole (1981) assessed the cognitive consequences of literacy, they "found localized changes in cognitive skills manifested in relatively esoteric experimental settings" (p. 234) instead of generalized changes in cognitive ability. With such problems of near transfer, the expectation of spontaneous transfer across diverse knowledge domains must be viewed cautiously. Nonetheless, we began our research enterprise with the hope of documenting such transfer from Logo programming, because of the novel character of the domain and the great interest such activities seemed to hold for students.

Apart from the important role of planning in programming, it is useful for many domains of activity to develop planning skills for the successful accomplishment of a coordinated set of goal-directed activities. A plan is a symbolic or literal representation of a set of actions designed to produce an intended outcome. Much human action appears to be guided by planning. Current general models of planning consist of four component processes that may be invoked recursively throughout the process of constructing and carrying out a plan: (1) the representation of the planning problem situation; (2) the process of constructing a plan to achieve that goal; (3) the execution of the plan; and (4) remembering the planning process. For example, the attempt to construct a plan may result in the redefinition of the goals, while trying to execute the plan may result in revision of the plan structure. Few developmental studies of planning have been carried out (e.g., Klahr & Robinson, 1981), although important aspects of these four interrelated components of the planning process for planning development have been discussed by Pea (1982), who reviews planning studies in cognitive science and artificial intelligence. Through this work we refined questions for studying the revisionary processes and products of planning activities in the two experiments reported here.

These two experiments focus on plan construction processes. In order to reveal how children plan, we were guided in our design of the tasks by empirical features of planning processes. Specifically, we felt the tasks should: (1) represent situations that are congruent with what is known about plan construction, especially when planning is likely to occur; and (2) provide "process accessible" data, externalizing the planning process to allow observers to see and record processes of plan construction in action.

With respect to (1), the planning context should: (a) be one where a child might be expected to see planning as appropriate and valuable; (b) be complex enough so that the means for achieving a goal are not immediately transparent and the possibility of alternative plans is recognized; and (c) involve a domain where children have a sufficient knowledge base so that action sequences can be planned and consequences of actions anticipated.

With respect to (2), the task should reveal: (a) whether alternatives are considered; (b) whether the planner tests alternatives by simulating their execution; (c) what kinds of revisions or debuggings of a plan are made; and (d) what different types and levels of planning decisions are made. Planning is appropriately characterized as a revisionary process. As a consequence of considering alternatives, effective planners revise their plans. They work between top-down planning strategies that create a plan from successively refining the goal into a sequence of subgoals for achievement in sequence, and bottom-up planning strategies that note the emergent properties of the plan or the planning environment and add data-driven decisions to the plan throughout its creation (Hayes-Roth & Hayes-Roth, 1979; Pea, 1982). Planners may go through many cycles of revision in considering the consequences of differently organized plans. In order to understand children's planning, a context should be provided for revealing these revisionary efforts. Are children able to revise their plans to take account of what they learn during plan construction? Furthermore, planning decisions can be made at different levels of abstraction. At an abstract level, one can think about design criteria for a plan (e.g., feasibility [Kotarbinski, 1965]) without considering specific actions or even a specific domain for the plan. Alternatively, planning can occur concretely as a sequence of local decisions without an overall framework. Such "planning in action" (Rogoff & Gardner, 1982) is probably more important in everyday actions than the creation of preformulated plans. The experimental task should be able to reveal these levels of plan decision making.

In addition to the features of plan construction, we considered the everyday practice of planning. To construct plans for situations requiring sequencing of multiple actions, planners often must simulate actions and observe their consequences. Although planning is frequently thought of as an internalized symbolic process requiring mental representation of and mental operation on symbolic elements, the complexity of symbolic manipulations for many planning tasks often leads to some kind of externalization of the planning problem space. For example, architects and interior designers may construct planning spaces in which they can physically try out alternate arrangements as part of the planning process, and programmers may create flowcharts.

Planning and Programming

How is planning manifested in programming? The core of computer programming is that set of activities involved in developing a reusable product consisting of a series of written instructions to make a computer accomplish some task. Global theories of expert programming skill acknowledge that programming is highly complex because "it involves subtasks that draw on different knowledge domains and a variety of cognitive processes" (Pennington, 1982, p. 11). As in the case of theories of problem solving in general, cognitive studies of programming reveal a set of distinctive mental activities that occur as computer programs are developed. These activities are involved throughout the development of a program, whether the programmer is novice or expert, since they constitute recursive phases of the problem-solving process in any general theory of problem solving (e.g., Heller & Greeno, 1979; Newell & Simon, 1972; Polya, 1957). They may be summarized as: (1) understanding/defining the programming problem; (2) planning or designing a programming solution; (3) writing a programming code that implements the plan; and (4) comprehension of the written program and program debugging. In Pea and Kurland (1983), we discuss each of these cognitive subtasks in detail; here we characterize only the roles of planning in programming.

After achieving an initial problem representation, and in the absence of a ready solution, the programmer must map out a program plan or design that will then be written in programming code. Atwood et al. (1980) provide an informative description of the requirements of this process:

> Software design is the process of translating a set of task requirements (functional specifications) into a structured description [design or plan] of a computer system that will perform the task. There are three major aspects to this description. The original specifications are decomposed into a collection of modules, or substructures, each of which

> satisfies part of the original problem description. This is often referred to as modular decomposition of the problem. In addition, these modules must communicate in some way. The designer must specify the interrelationships and interactions among the modules [also called procedures in smaller systems]. This includes the control structure, which indicates which modules are used by a given superordinate module to do its work and the conditions under which they are used. Lastly, a design may include a definition of the data structures required. (p. 3)

According to Brooks (1982), one-third of the time a program team spends on a software project (including coding and testing) should be devoted to planning. Atwood et al. (1980), in a detailed analysis of the think-aloud protocols of two expert software designers as they solved a design problem, found that they had available many general plan-design strategies, such as problem decomposition, subgoal generation, retrieval of known solutions, generation and principled (or policy-driven) selection of alternative solutions, and evaluative analysis and debugging of solution components.

One may raise the objection that it is possible to bypass planning in program development; that is, one may first make an initial reading of the problem and then compose code at the keyboard to achieve the task. And it has been observed (Galanter, 1983) that preplanning frequently occurs when programming in PASCAL (a compiler language), but often little or no planning occurs prior to writing code for programming languages such as BASIC or LOGO (interpreted languages). What are we to make of this observation in terms of defining planning as a distinct cognitive subtask in programming? Is it optional? The answer to this question certainly has consequences for thinking about the cognitive outcomes of programming.

The distinction commonly made between preplanning versus planning-in-action (Rogoff & Gardner, 1983) is important. Programmatically, our planning tasks should be sensitive to both. The BASIC programmer is planning-in-action as he or she generates programming code without a prior plan, making decisions that are guided by the constraints of the program content. Bamberger and Schon (1982) have described such planning-in-action creative processes in art, music, and other related domains as a consequence of an iterative series of "conversations" between the creator and his or her uncompleted creations. Bereiter (1979) has characterized a similar process in composing language text as "epistemic," in which one comes to see and understand new things about what one wants to express as ideas are channeled into a written product.

Although planning-in-action is certainly possible, even sufficient, to produce some programs, such planned-in-action programs create problems for the inexperienced programmer. In contrast, expert programmers can draw on their knowledge of a vast range of plans when creating programs (Atwood et al., 1980; Soloway et al., 1982). The novice programmer who wishes to revise a program that he or she created without preplanning will undoubtedly find it difficult to understand and, therefore, to debug it. Expert programmers need do little in the way of preplanning because of the automaticity of many programming projects, and because they have had a great deal of experience with similar programs or software systems. In other words, the expert programmer is able to integrate the subtasks of planning and code writing. In contrast, the child or novice programmer has neither the sophisticated understanding of programming code nor the experience of devising successful programming schemas necessary for engaging in planning-in-action.

Taking all these considerations into account, we decided that a classroom chore-scheduling task, analogous to a planning scenario used by Hayes-Roth and Hayes-Roth (1979), met this series of requirements for a planning task. We found from classroom observations that all children had to carry out certain classroom chores on a regular basis. The children were familiar with a list of chores (e.g., washing the blackboards, watering the plants) and the actions involved in doing each. The task was made novel by requiring children to organize a plan that would allow one person to accomplish all the chores. We designed a classroom map as an external representational model to support and expose planning processes.

Study One

Participants

Thirty-two students from a private school in Manhattan participated in the study. The children were seen in two sessions, at the beginning and end of the year, the second session occurring four months after the first. Half of the children were 8- and 9-year-olds (mean age, 9.3 years; s.d., 0.7 years); the other half were 11- and 12-year-olds (mean age, 12 years; s.d., 0.6 years).* These two age groups were selected not for theoretical reasons, but because their teachers were willing to participate. Nonetheless, this age range is representative of children learning Logo programming in American schools. The 32 children came from four different classrooms. The experimental

*Children's ages at the time of the first session.

groups were composed of four boys and four girls of each age who were learning Logo computer programming; the control groups consisted of four boys and four girls of each age not receiving the treatment.

The selection of the children was not random. Participants in the experimental groups were selected on the basis of two criteria: (1) a large amount of time working with Logo during their first two months of computer use (prior to the first experimental session); and (2) teacher-assessed reflectiveness and talkativeness so that rich think-aloud protocols would occur during the task.

Only the second criterion was used for the control groups. To ensure comparability of groups, we also administered a digit-span task and the WISC Block Design subtest. The former was used to assess the size of a basic processing capacity, the latter to determine cognitive style in terms of field-dependence or field-independence (e.g., Case, 1974; Case & Globerson, 1974). Experimental and control groups for both ages did not significantly differ on scores for either measure.

Materials

A transparent plexiglass map of a fictitious classroom was developed for the task (see Figure 1). The map was 22" by 30", with a scale of 1" to 15". For the second task session, a replica map was used that was different only in that its orientation was transposed 180° to ensure that children would not remember their specific movements or direction of the spatial approach from the first session.

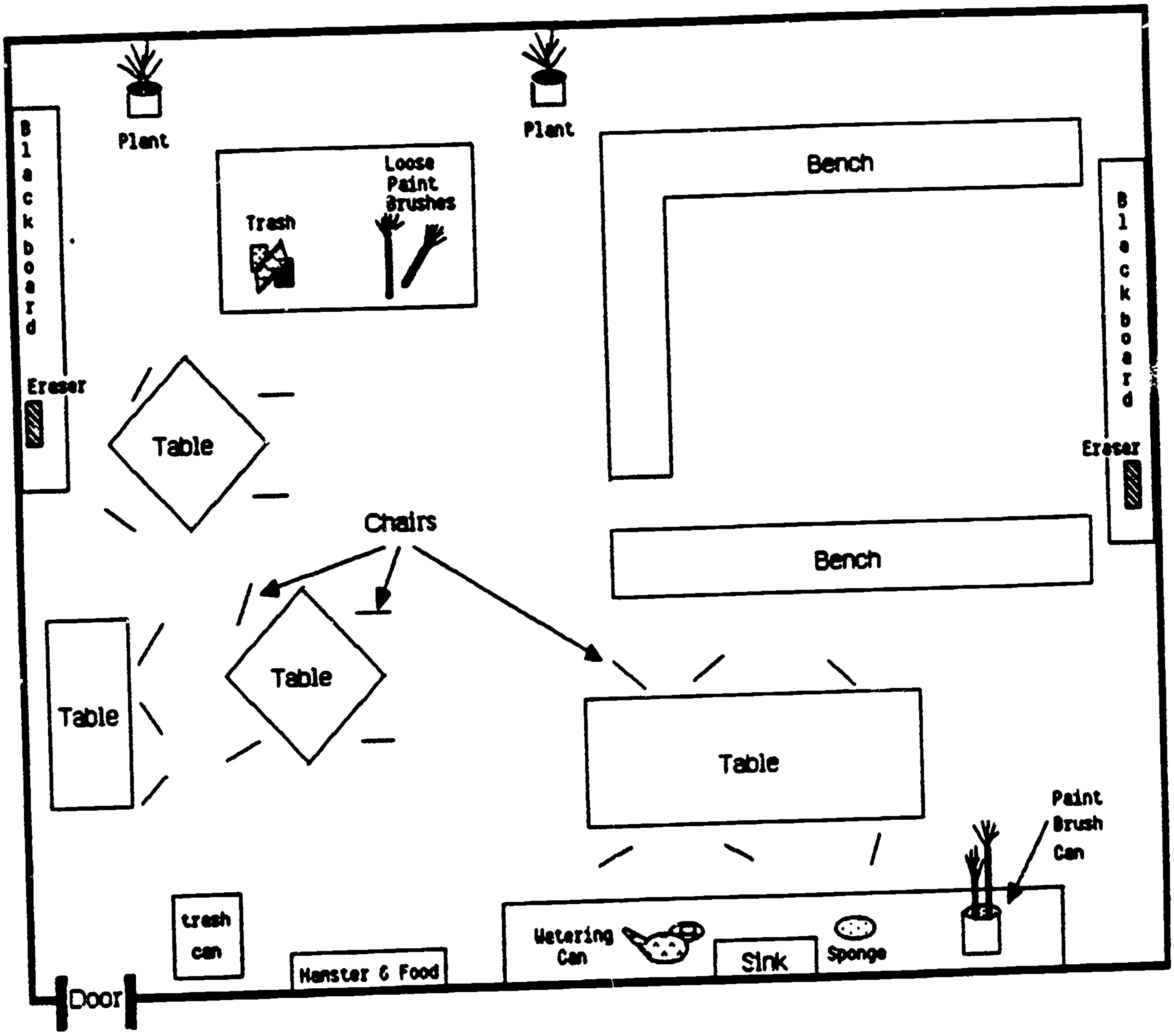
There were six major chores: (1) watering (two) plants; (2) erasing and washing (two) blackboards; (3) feeding a hamster; (4) putting each of 17 chairs under its adjacent table; (5) washing (five) tables; and (6) putting away objects (returning and washing paintbrushes; disposing of trash paper) lying on the art table. These chores may be accomplished with a minimum of 39 distinct chore acts. Some of the chore acts are subgoals since they are instrumentally necessary to accomplish others (i.e., the watercan is needed to water plants; the sponge is necessary for washing tables and blackboards). Finding the optimal sequencing of these chore acts is thus a challenging task.

Experimental Design

The planning task was administered in both sessions one and two. The two sessions were separated by four months (early and late in the school year) during which time the children were learning Logo computer programming. Between-participant group variables were:

Figure 1

Diagram of Classroom Model - Study One



(1) Group (Logo, no-Logo); (2) Sex (male, female); and (3) Age (younger, older). The key within-participant variables were: (1) scores for first-last plans within session; and (2) scores for session one versus session two.

Procedure

Logo classroom context. The study was located in two classrooms at a private school in Manhattan. One classroom included 25 (11 boys, 14 girls) 8- and 9-year-old children (third and fourth graders); the other consisted of 25 (11 boys, 14 girls) 11- and 12-year-old children (fifth and sixth graders). The children encompassed a variety of ethnic and socioeconomic backgrounds and a range of achievement levels. Many of the children were, however, above national norms in school achievement and came from upper-middle-class and professional families. The experimental groups were students in these two classrooms; the control groups were made up of students in same grade-level classrooms in the same school.

Each classroom had six microcomputers during the 1981-1982 school year. Both the younger and older groups had three Apple II Plus computers and three Texas Instruments (TI) 99/4 computers. In each class, children were learning Logo, a programming language designed to be easily accessible to children and to encourage the development of thinking skills (Papert, 1980; Byte, 1982). A widely distributed MIT Artificial Intelligence Laboratory version of Logo software was used for the Apple computers; a commercially available TI version of Logo was used for the TI computers. Teachers received extensive training in Logo classroom use before the school year, and the computer programming activities during the year were intended by the teachers to be largely child-initiated so as to encourage the Piagetian discovery-learning pedagogy advocated for Logo by Papert (1980). While teachers gave the children some simple instruction in Logo during the first several weeks and occasionally held group sessions to introduce new aspects of Logo during the year, their self-defined role was principally that of responding to students' questions and problems as they arose. Students were encouraged to create and develop their own computer programming projects.

Teachers scheduled computer use for students in their classrooms so that everyone would have equal time to use them--about two 45-minute work periods per week. There were additional optional times for computer use throughout the day--before school and during lunch period when computers were available on a first-come, first-served basis. Logs kept at each computer over the course of the year showed that, on the average, the children spent about 30 hours programming in Logo.

Classroom chore-scheduling task. The children were tested individually. E took the child from the classroom to a filming room, where he or she was seated at a table with the plexiglass map upright on an attached stand 6" away and tilted back 6° from the vertical plane. A videocamera located approximately 8' from the map filmed each session through the map.

The child was told that the goal would be to make up a plan to do a lot of classroom chores. The loci of the chores on the map were pointed out as the chores were described. E then explained that the child should work to come up with the shortest spatial path for doing the chores, and that he or she could make up as many plans as were needed to arrive at the shortest plan. The child was then instructed to think out loud while planning, and to use a foot-long wooden pointer to show the path taken to do the chores. Finally, the child was given a pencil and paper to make notes (optional and rarely used), and a list of the six chores to keep track of what she or he was doing.

After the child had indicated completion of a plan by moving the pointer to the exit door on the map, E asked: "Can you make up a shorter plan?" If the child answered "yes," the session continued with the formulation of another plan; the session terminated when the child believed he or she had arrived at the shortest plan he or she could formulate. The same procedure was followed for the second session four months later.

Results

Three principal types of analysis were performed. In the first section, we review analyses of plans considered as products, with a focus on efficiency. In the second section, we consider the types of plan revisions children made, that is, what were the qualitative features of the plans that contributed to plan improvement? In the third section, we examine the planning process, especially in terms of the types and levels of abstraction of component decisions of the planning process. In the final section, we integrate the findings for these analyses by examining the extent to which a child's processes of plan formulation contribute to the quality of their plans. Unless otherwise specified, differences are statistically significant for alpha less than .01 and interactions are not significant. There were no sex differences revealed for any comparisons.

Product Analysis: Plan Efficiency

Data reduction from the videotaped sessions took place in three phases. First, the videotapes were carefully transcribed, with sequential notations made of utterances, pointing, and other gesturing. Then, for analyses of plans as products, we recorded the sequence of chore acts (moves) for each plan created. Finally, for every plan created, we calculated the distances that would be traversed if the plan were to be executed.[1] The mean number of plans per child was 3.64, and there were no significant group or age differences in this respect.

The analysis of plans as products involves the distances of the individuals' plans, and their efficiency relative to the shortest distance plan for accomplishing the chores. The key variable for efficiency analyses is "plan route efficiency," calculated as a score (as in Goldin & Hayes-Roth, 1980):

$$\text{Route efficiency} = 100 - \frac{(\text{Total distance} - \text{Optimal distance}) * 100}{\text{Optimal distance}}$$

We believe the route efficiency score represents the single most straightforward index of the effectiveness of an individual's planning efforts in this task. Since not all children created the same number of plans, we use the first and last plans for analyses.

Route efficiency score significantly increased with age, from first to last plan within session, and for the mean session route efficiency score across age groups, which increased from 65 to 80 (out of 100 possible). A better sense of the improvement of scores from first to last plan within and across sessions may be gleaned from Table 1, which is displayed in Figure 2. The Logo programming group did not differ from controls for route efficiency scores, at either age, for any plan or session. Further analyses for both sessions reveal that, compared to the younger group, the shortest plans of the older group were shorter, as were their longest plans. No comparable differences were found between experimental and control groups. Neither WISC nor digit span were significantly correlated with route efficiency scores.

Qualitative Analysis of Plan Improvement Through Revision

We have shown that each age group improved in efficiency from first to last plan, but how did plan revisions lead to improvements? We wanted to know what kinds of plan revisions were made, so we needed fine-grained observations that would point to concrete features that varied across plans, rather than, in the case of strategy analyses, descriptions of general task approaches which might map somewhat indirectly onto concrete plan features. We will refer to this approach as a featural analysis.

Table 1

Mean Plan Route Efficiency Scores Across Plans and Sessions (Study One)

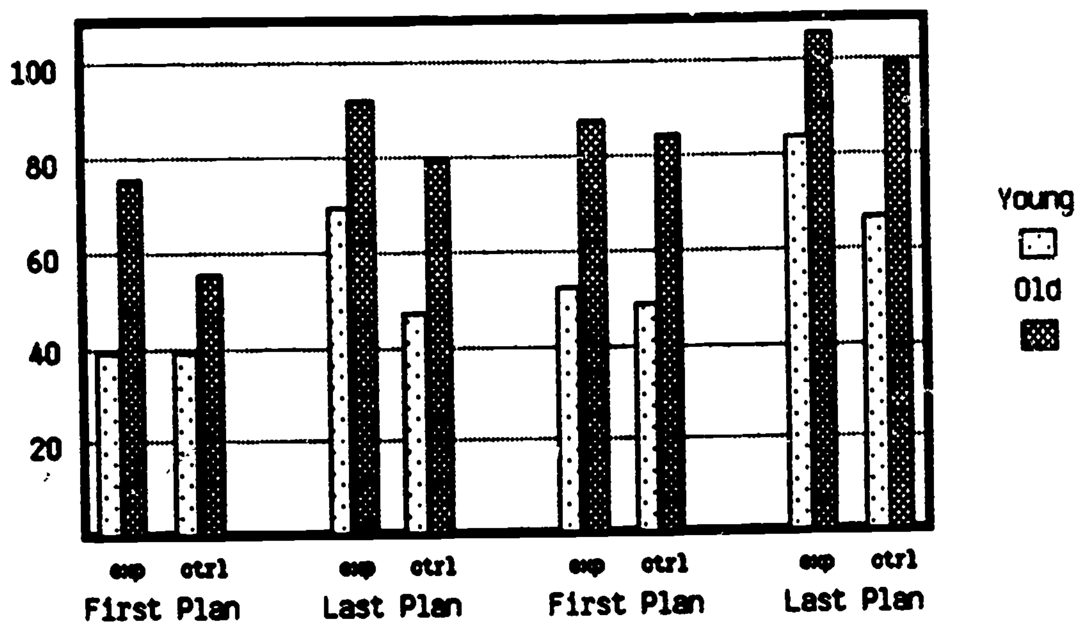
	Session 1				Session 2			
	1st Plan		Last Plan		1st Plan		Last Plan	
	mean	sd	mean	sd	mean	sd	mean	sd
Logo Programmers								
Younger	39.6	(19.6)	70.1	(23.8)	52.7	(37.1)	86.4	(14.6)
Older	76.3	(26.9)	96.2	(12.1)	89.3	(25.3)	107.3	(5.5)
Nonprogrammers								
Younger	39.3	(26.3)	46.8	(32.3)	49.5	(26.6)	67.4	(28.3)
Older	55.5	(42.9)	80.5	(23.2)	85.2	(19.1)	100.0	(7.4)

Our aim was not to examine all types of plan revision, but only those accounting for the bulk of progress made across plans. We derived such a set by observing many plans and noting the major changes in plan structure that led to improvements. For the most part, we can characterize the children's substantive revisions of structure to improve their plans as resulting from "seeing" the chores differently over time. These phenomenological shifts, whereby the task and its elements come to be understood differently from plan to plan, are characteristic of human problem-solving efforts, and are an aspect of problem-solving skill that might be expected to improve as a consequence of learning to program. The general importance of such "reseeings" while solving problems has been extensively documented by Gestalt psychologists (Wertheimer, 1961) and recent studies of problem solving in cognitive science (e.g., Bamberger & Schon, 1982; DiSessa, 1983; Heller & Greeno, 1979).

More specifically, the initial formulation of our task as the carrying out of a set of named chores (e.g., "cleaning tables," "washing blackboards," "pushing in chairs") is a frame or set for problem understanding that must be broken for the task to be accomplished effectively. Performing each named task, in whatever order, is not

Figure 2

Plan Route Efficiency - Study One



Session One Session Two

-n effective plan. Each chore must be decomposed into its component acts, and the parts must then be reconstructed and sequenced into an effective all-encompassing plan. The child's understanding of part-whole relations for the task is thus transformed during plan revision. To move toward the optimal solution of this planning problem, a child must reconfigure the chore "chunks" in terms of their spatial distribution on the classroom map. Major breakthroughs in plan structuring occur through discovering spatial clusters of chore acts. Progress in plan structure is thus made through restructuring the chunks of activities to be accomplished--from a list of named chores to a list of spatial clusters of chore acts.

The kinds of changes children made are better understood in this context. There are two major types of plan features, and we have assigned one point for each of 14 plan features present. There are nine "chore act clusters," and five plan features that involve "movables" (such as brushes, watercan, sponge). In all cases, the plan feature eliminates redundancies in travel that arise when an area in the classroom is visited twice to do different chore acts that could be accomplished in one trip. Below, we illustrate the types with an example of each (details are provided in footnote 2). For example, for one chore act cluster, an improved plan occurred when each of the tables with chairs was visited only once, at which time the table was washed <u>and</u> the chairs at that table were pushed in, rather than performing <u>each</u> of these tasks in two separate trips. For one cluster type involving "movables," major improvements in plan structure occurred when the sink was visited only after all three movable things to be returned to that location (sponge, watercan, paintbrushes) were no longer needed.

Children's plans were then analyzed in terms of these plan features. We found that mean cluster scores and plan efficiency scores, discussed earlier, were highly correlated for first and last plans for both sessions (<u>r</u>s ranged from .66 to .72). The qualitative analysis of plan clusters was thus related to the quantitative measure of plan distance. More efficient organization of chore acts into clusters was highly correlated to shorter plan distance.

The mean plan cluster score significantly improved for each age group across plans and sessions, but Logo programmers did not differ from the control groups on any of these comparisons. The children reorganized their plans into more efficient clusters during the revision process whether or not they had programmed.

<u>Process analyses</u>. We also wished to compare planning processes across children and plans. In creating their plans, did our Logo programmers engage in more advanced decision-making processes than

the nonprogrammers, even though their plans were not more efficient? Work by Hayes-Roth and colleagues has led to a detailed system for coding the levels of abstraction and types of planning decisions made by adults as they think aloud while constructing plans to carry out errands in an imaginary small town. Our categories are a subset of those used by Goldin and Hayes-Roth (1980) and Hayes-Roth and Hayes-Roth (1979) for categorizing planning decisions made by adults, supplemented by categories that emerged for this classroom environment. We examined the process of plan construction by categorizing each segment of the children's think-aloud protocols in terms of the type of planning decision being made and its level of abstraction. Relevant aspects of the system for our purposes will now be briefly reviewed.

The type categories specify different conceptual categories of decisions made during planning. The first three types of decisions choose plan features; the other two are more strategic in nature, determining features of the planning process:

1. Plan: represent specific actions the planner intends to take in the world (e.g., "go to wash the art table this way," while tracing out a path).

2. Plan abstraction: select desired attributes of potential plan decisions, noting kinds of actions that might be useful without specifying the actual actions (e.g., "go to closest chore next" or "organize plan around bunches of chores").

3. World knowledge: assess data (e.g., of chore or instrument locations, distance, or time) concerning relationships in the task environment that might affect the planning process (e.g., "the hamster is next to the door" or "the chores are all in a circle").

4. Executive: determine allotment of cognitive resources during planning, such as what kinds of decisions to make first, or what part of the plan to develop next (e.g., "I'll decide what order to do the chores in before figuring out how to walk").

5. Metaplan: reflect planner's approach to the planning problem and the methods he or she intends to apply to it, or establish criteria to be used for making up and evaluating prospective plans.

A complementary analysis of planning decisions codes their level of abstraction. For "abstractness level" categories, decisions at each more specific or concrete level specify a more detailed plan than those at the next higher level of abstraction. Levels for all types except the "metaplan" are hierarchically organized (see Appendix for details). A good planner moves flexibly among both types and levels of abstraction as she or he constructs a plan (Goldin & Hayes-Roth, 1980). Here we present an example of level stratification for the "plan" type category described earlier, moving from abstract to concrete down the list:

Level	Definition
1A: Outcome	Determine which chores will be accomplished when plan is executed (e.g., "I'll definitely do the hamster and the plants, because they'll die").
1B: Design	Determine specific spatiotemporal approach to planned activities (e.g., "I'll do the chores by going in a circle").
1C: Procedures	Determine specific sequences of gross actions (e.g., "I would do the hamster, and then get the sponge," without noting path).
1D: Operations	Determine specific sequences of minute actions (e.g., noting the details of the path for a sequence of gross actions in the plan).

With the process analysis, we ask whether the organization of the planning process in terms of the types, levels, and sequences of planning decisions is different for the programmers than for the nonprogrammers. Below we survey findings on: (1) frequencies of different types of planning decisions; (2) decision choice flexibility; (3) relationships between the amount of "executive" and "metaplanning" activity during the planning process and decision-choice flexibility; and (4) whether students' scores on cognitive style and processing capacity measures distinguished different planning process profiles.

1. Frequencies of Planning Decisions in Terms of Types

Five types of planning decisions were distinguished. The first three types--Plan, Plan Abstraction, and World Knowledge, referred to as "low level"--concern specific details of planning. The latter two--Executive and Metaplan, referred to as "high level"--pertain to higher level executive or metacognitive aspects of plan decision-making.

Each student's protocol was divided into segments assumed to represent individual planning decisions. The mean number of segments for all plans produced was 43.8, which did not significantly differ across groups, plans, sessions, or by age. Most of these planning decisions were of the Plan type (Session 1 [S1]: 95.7%; Session 2 [S2]: 96.4%). The overall frequencies of decisions of other types were Plan Abstraction (S1: 0.6%; S2: 0.5%), World Knowledge (S1: 1.6%; S2: 1.4%), Executive (S1: 1.7%; S2: 1.2%), and Metaplan (S1: 0.4%; S2: 0.5%). "High level type" planning decisions constituted about 2% of all the planning decisions the children expressed. The Logo programming group did not differ from the control groups on any of the comparisons for types of planning decisions. Nonetheless, we found interesting differences in when and by whom such higher level decisions were made.

As for differences in the types of planning decisions made for first versus last plans by the 32 children, only for session one did we find that children made significantly more high level-decisions in their first plans than in their last, and older children produced more high-level decisions than did younger children. These plan and age effects dissipated for the second session.

2. <u>Decision-Choice Flexibility</u>

As in planning studies by Goldin and Hayes-Roth (1980), we tried to determine the degree of flexibility of a child's decision making during the planning process in two ways: (1) by looking at the number of transitions a child made between types of decision making while creating the plan, and (2) by looking at the number of transitions made between levels of decision making irrespective of the decision type. For both sessions, the mean number of type transitions per plan is highly correlated with the mean number of level transitions per plan. The programmers did not differ from the nonprogrammers on these indices of decision-choice flexibility.

The mean number of type transitions for the group of 32 children was 2.4. More type transitions were made in the first than in the last plan, but not significantly so. In session one, older children made significantly more type transitions per plan (4.0) than did younger children (1.5).

The mean number of level transitions for the group of 32 children was 2.7, not differing by plan or by session. For both sessions, older children made significantly more level transitions per plan (4.0) than did younger children (1.4).

3. High-Level Planning Decisions and Decision Flexibility

We find that children who engage in more high-level decision making during planning (i.e., types 4 and 5) also display more flexible decision making by shifting opportunistically between different decision types and levels. High-level decision making during planning was significantly correlated with both the number of type and level transitions. Goldin and Hayes-Roth (1980) report similar findings for adults in their errand-scheduling task.

4. High Level Planning Processes, WISC, and Digit Span

Neither processing capacity (as indicated by digit span) nor cognitive style (as indicated by WISC Block Design score) were significantly correlated with the frequency of high-level planning decisions. Although the mean number of type transitions was significantly correlated with digit span, there were nonsignificant correlations of .3 to .4 between digit span and mean number of level transitions. WISC score did not significantly correlate with either mean number of type or level transitions per plan.

Relating Plan as Product and as Process

How related are the decision-making processes to the effectiveness of the plan as a product? We find that, for this task, the process and product measures are weakly related. Neither the plan efficiency mean score for all plans produced nor the distance of the shortest plan a child created correlated significantly with any of the high-level plan process measures; that is, mean number of type transitions per plan, mean number of level transitions per plan, or frequency of high-level (types 4 and 5) planning decisions.

We also tested for a relationship between the frequency of high-level planning decisions and mean cluster scores from the feature analysis. The nonsignificant relationships indicate that children revise their plans to accomplish the acts more efficiently without necessarily using (verbally explicit) metaplanning resources. Only for the last plan of the younger children in the first session are these variables significantly correlated ($\underline{r} = .65$).

On the face of it, these results suggest that a school year of Logo programming did not have a measurable influence on the planning abilities of these students. While we grant that an average of 30 hours of programming is small compared with what professional programmers or college computer science majors devote to such work, it is a significant amount of time by elementary school standards. If we take seriously the claim that writing Logo programs has a positive effect on children's high-level thinking skills, we would expect to see some effect from this amount of effort.

Study Two

Since, in our second experiment, we wanted to look more closely at potential programming effects on planning skills, we designed a new task that resembled programming in its deep structural features (e.g., on-line feedback on the success of planning efforts). These alterations were constructed with the expectation that, given the closer analogs between the demands of this task and those of computer programming, such planning skills that students might have developed for the latter domain would more likely be transferred to the new task context.

Participants

The second experiment took place one year after the first, in the same school and with the same two teachers. As in the first experiment, half of the children were 8- and 9-year-olds (from combined classes of third-fourth graders), the other half were 11- and 12-year-olds (from combined classes of fifth-sixth graders). A major difference from the first year was that both teachers decided to take a more directive role in guiding their students' explorations of Logo. In the classroom with the younger students, this amounted to the teacher's giving weekly group lessons and demonstrating key computational concepts and techniques. The older students were also given more group lessons and required to complete specific assignments. In both classrooms, however, students were encouraged to develop their own projects when working on their own at the computer.

Study two followed the same general pre-post design of study one. The first testing session took place early in the school year, followed by a second session six months later. Thirty-two students participated in both sessions of study two (none of whom had participated in the first study). For the second session only, an additional 32 students were tested in order to produce a more sensitive test for the planning task analyses. The 32 children tested for the first session were drawn from four different classrooms (two combined third-fourth grades; two combined fifth-sixth grades). Four boys and four girls of each age formed an experimental group that was learning Logo computer programming, and four boys and four girls of each age constituted a control group not receiving the treatment. For the second session, an additional 32 students were drawn from these same classrooms. Again, four boys and four girls of each age were selected. Participant selection in both cases was random, with the stipulation that equal numbers of boys and girls be selected from each classroom for each of the experimental and control groups.

Materials

Two different versions of the chore-scheduling task were used in this study. The first was identical to the chore-scheduling task used in study one, and was administered early in the school year. For the second session, a new computer-based, chore-scheduling task was developed. This task incorporated new design features that made the task bear a closer resemblance to programming. Modifications to the original task were also made to eliminate problems with scoring that resulted because students developed a number of different plans, or plans that were incomplete. In addition, the new task was designed to monitor and record automatically each student's performance. This enabled us to dispense with the use of videotaping to score each student's performance. And because the computer could provide them with immediate, meaningful feedback on the adequacy of their plans after each attempt, students were able to monitor their own performances.

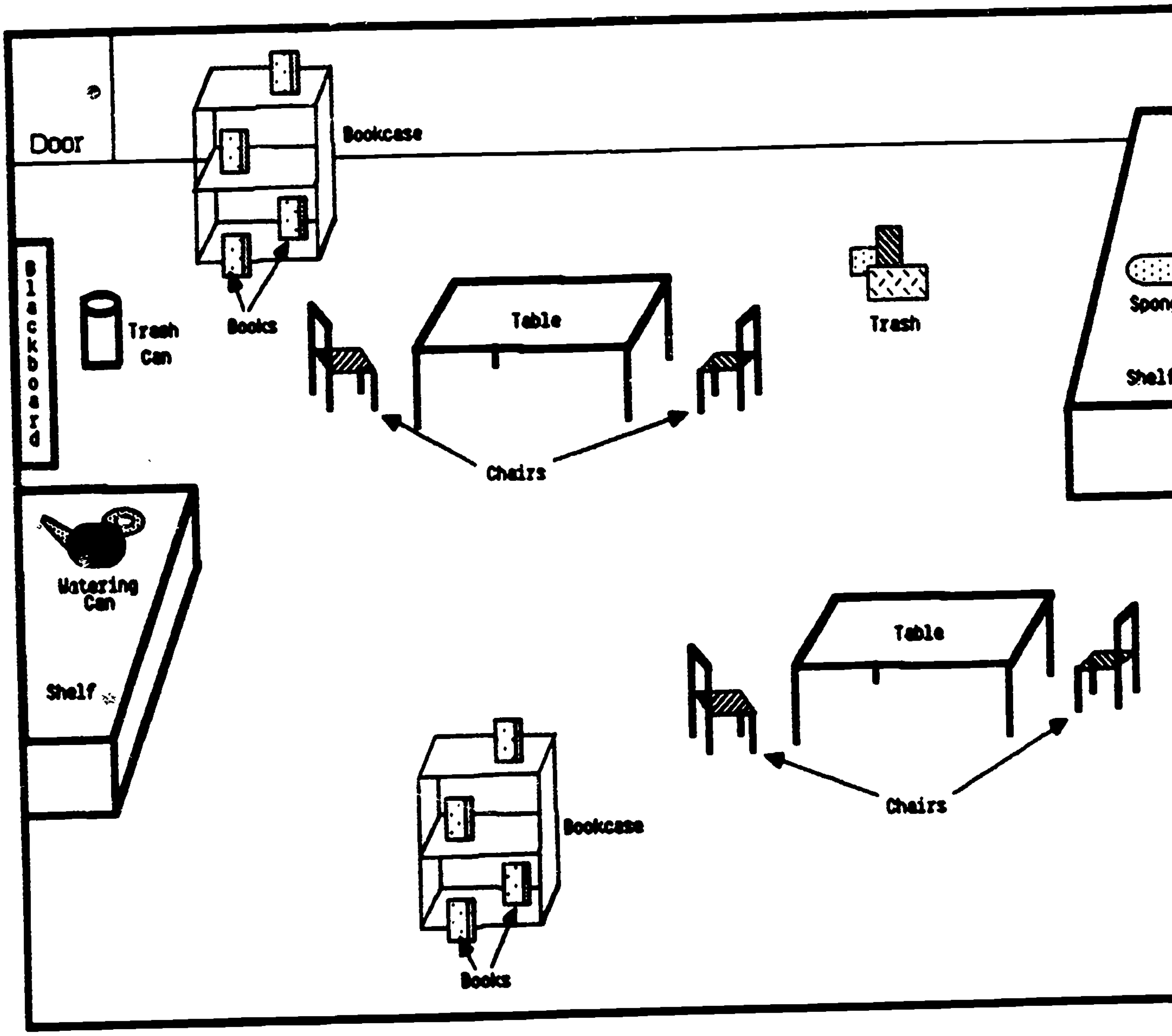
The new task consisted of four components: (1) an 8½" by 11" colored diagram of a classroom; (2) a set of six 4" by 6" goal cards, each depicting one of the six chores to be accomplished (wiping off the tables, wiping off the board, watering the plant, straightening the books, pushing in the chairs, and throwing away the trash); (3) a plan input interface, namely, a microcomputer program that enabled students working with the experimenter to design and check their plans interactively; and (4) a graphics interface that enabled students to see their plans enacted in a realistic representation of the classroom (see Figure 3).

The plan input and graphics interfaces were computer programs that created a graphics robot programming and testing environment, or microworld, within which children could develop their plans. With these interfaces, the children could "program" a robot using a simple, English programming language, and then see their plan actually carried out.

The commands in the robot programming language (RPL) consisted of a set of six actions (walk to. pick up, put down, wipe off, water, straighten up), and the names for all the objects in the classroom. A typical fragment of an RPL program is shown in Table 2. Each action-object pairing constituted a move in the plan. Each action and each object were coded to a single key on the computer keyboard.

Figure 3

Diagram of Classroom Model - Study Two



As the student talked through a plan while looking at the classroom diagram and goal cards, the experimenter keyed in each move using just two keystrokes--one for the action and one for the object. If the student gave a command that could not be carried out at that point in the plan (e.g., telling the robot to wipe off the table before going to pick up the sponge), the computer program immediately rejected the move and provided a precise context-specific error message on the screen (e.g., I'M NOT CARRYING THE SPONGE). If a student indicated that his or her plan was done when there were actually one or more chores still remaining, the program provided a message to this effect, and a list of the outstanding chores appeared on the screen. A message always displayed on the screen informed students that they could at any time ask to see the list of remaining chores or review their plan by having it listed on the screen. Together, these features ensured that all the students would develop runnable, albeit not necessarily optimal, plans.

Table 2

Example Fragment of an RPL Command Program

```
WALK TO THE SHELF
PICK UP THE SPONGE
WALK TO THE BOARD
WIPE OFF THE BOARD
WALK TO THE WATERING CAN
PUT DOWN THE SPONGE
PICK UP THE WATERING CAN
WALK TO THE PLANT
WATER THE PLANT
WALK TO THE SPONGE
PUT DOWN THE WATERING CAN
PICK UP THE SPONGE...
```

The second part of the new classroom chore-scheduling task was a graphics interface designed to provide detailed feedback to the student on the adequacy of his or her plan. There were four types of feedback: (1) a readout of the total time the student's just-completed plan would take if carried out in action; (2) a representation of a classroom displayed on a high resolution screen in which a step-by-step enactment of the student's plan could be carried out under the student's control; (3) a step-by-step readout of each move the student had entered and the time it took the robot to carry out each move; and (4) a hard-copy printout of the student's plan that could be referred to during subsequent planning attempts.

Experimental Design

The original planning task was administered to 32 students at the beginning of the school year. For the next six months, half of these students learned Logo computer programming. At the end of the school year, the original 32 students plus a group of 32 additional students (half of whom had also been learning Logo) were administered the new planning task. Between-participant grouping variables were: (1) Group (Logo, no-Logo); (2) Condition (feedback, no-feedback); and (3) Age (younger, older). For the pretest, the key within-participant variables were the same as those in study one. The key within-participant variables for the second session were: (1) total time for the robot to carry out each of the three plans; (2) total time for thinking about what move to make next in each of the three plans; (3) use of feedback and debugging aids in each of the three plans; and (4) degree of similarity of each plan to the others.

Procedure

For the first session, the procedure was identical to that in study one. In the second session, children were taken individually from their classrooms to a testing room, where they were seated at a table in front of two computer monitors (one color and one monochrome) connected to an Apple computer. The diagram of the room, with the six goal cards and a picture of the one-armed robot laid out around it, was placed on the table directly in front of the monitors.

Students were told to imagine that they had a robot who could understand and carry out commands to perform classroom duties. Their task was to devise a plan for the robot to clean up a classroom in the least possible amount of time, covering the shortest possible spatial path. Students were told that the robot would follow their instructions to the letter, and that the robot did the chores rapidly but moved very slowly. Therefore, to make a good plan, they should try to minimize the total amount of the robot's travel in the room. To emphasize the need to develop an efficient plan, the experimenter told each student that the best possible plan was one in which the robot would complete its task in approximately 15 minutes. Students were told that each action the robot could carry out would take a constant amount of time (e.g., picking up or putting down an object equalled 15 seconds) and that the robot moved at a constant speed wherever it went. They were further informed that the robot had one arm, and therefore could pick up and carry only one object at a time (e.g., the sponge or the watercan, but not both). To help students remember all its characteristics, a drawing of the robot with its one arm and small wheels was kept alongside the goal cards in front of the students at all times.

As the chores were described, students were shown the goal cards and the loci of the corresponding chores on the room diagram. The experimenter then explained that the students should work to come up with the shortest spatial path for doing the chores, and that they would have the opportunity to create three plans in order to arrive at the shortest one possible.

After these instructions were offered, the experimenter pressed a key to start the program running and initialize the millisecond computer clock. The purpose of the clock was to record the intervals between the student's moves (thinking time). This enabled us to determine how reflective each student was while creating each plan, and where in the planning process the students spent more time thinking.

Students were given as much time as they needed to think about what to do and to call out each individual move. The experimenter typed each move into the computer, where it was either accepted and added to the plan list, or immediately rejected and the student told what was wrong. The computer did all the monitoring and error checking, and gave the only feedback the child received on how the plan was developing. The experimenter's role was to explain the purpose of the task, and then serve as the student's typist without commenting on the adequacy of individual moves or plans. When all the chores were completed and the robot was directed out of the classroom door, the program calculated and then displayed how long the just-entered plan would take. This estimate was based on the set amount of time each chore act took to carry out, plus "travel time" calculated by converting into time units the distance the student's plan required the robot to travel.

In the feedback condition, as soon as a student completed a plan, a color image of the classroom appeared on the graphics screen and, simultaneously, the student's first move was printed on the text screen. The student was given a hand-held button that, each time it was pressed, took the program through the plan one move at a time. As the student pushed the button, a line corresponding to the move entered was drawn on the room diagram indicating the path the robot would follow in carrying out the student's plan. Simultaneously, on the text screen, the specific move the student had entered was printed out (e.g., WATER THE PLANT), as was an elapsed time counter indicating the total time needed by the robot to carry out the plan up to the current move (again based on the distance traveled plus the time needed to do each of the indicated actions). Simultaneously, the student's plan (minus the commands to check or review the plan) was printed out on a printer so that, when devising subsequent plans, students could see exactly what they had done on their earlier attempts.

In the no-feedback condition, students created three plans in a row without benefit of seeing them enacted or printed out. Upon the completion of each plan, they were simply told how long it would take the robot to carry it out, and then went directly to entering their next plan.

For each move of each plan the computer recorded four kinds of data:

- What the move was (e.g., WALK TO THE PLANT).
- Whether the move was legal (e.g., if the command WATER THE PLANT was issued before telling the robot to pick up the watering can, the move was recorded, but flagged as illegal so it would not be added to the plan listing that the student could see, and was not attempted by the robot when it carried out the student's plan).
- How long the move would take the robot to carry out.
- How long the student paused to think between the previous move and the current one.

These raw data were used to create the following set of plan-level variables:

- Total time: the total time the robot would take to enact the plan.
- Illegal moves: the number of times the student issued a command that the robot could not carry out.
- Check list: the number of times the student asked to check the list of remaining chores.
- Review plan: the number of times the student asked to see the listing of the plan up to its current position.
- Total thinking time: the total amount of time that elapsed between each move.

- Thinking time by position: the average amount of time the student paused to think between moves in the beginning, middle, and end of the plan.

- Pauses: the number of times the student paused for 10 seconds or longer between moves in the plan.

Also, the similarity of each plan to the other two was calculated in the following manner. First, each set of three consecutive moves (1-2-3, 2-3-4, 3-4-5, etc.) in the first plan was compared to each set of three consecutive moves in the second plan. Wherever three moves coincided (e.g., moves 3-4-5 in plan one were the same as moves 13-14-15 in plan two), a cluster of size three was counted. When all the clusters of size three had been checked, each set of four consecutive moves in the first plan was compared to each set of four consecutive moves in the second plan. For each match, a cluster of size four was counted. This process continued until sets of all possible sizes in plan one (i.e., from sets of size 3 to the total number of moves in the plan) had been compared to all possible sets of the same size in plan two. Once this was done, the process was repeated, comparing plan two to plan three, and then all three plans to each other.

In order to calculate the degree of overlap or similarity between plans, the number of clusters of each size was adjusted to reflect the fact that a cluster of size X was also two clusters of size X-1, three clusters of size X-2, and so on. Thus, for every cluster of size five that was located, two clusters of size four and three clusters of size three were subtracted from their respective totals. In this way, the cluster counts that were finally left reflected the unique number of sets of each cluster size that overlapped between plans.

Our measure of plan similarity, or percent plan overlap, between one plan and another was calculated by multiplying the number of clusters of each size by cluster length, summing these totals, and dividing by the total number of moves in the shorter of the plans being compared.

Results

The logic of the analyses for study two was somewhat different from that of study one. In study one, we were interested in characterizing students' overall planning processes, and the qualitative features that contributed to plan improvement. Since study one had failed to distinguish between students on the basis of their programming background, study two was designed to take a closer look at potential programming-related effects.

The original version of the chore-scheduling task was administered as a pretest near the beginning of the year to verify the comparability of the treatment and control groups. The plan-route efficiency scores on this task were subjected to an analysis of variance in which Age (older, younger) and Group (Logo, no-Logo) were the independent factors. This analysis indicated that, as in study one, there was a highly significant main effect for Plan ($\underline{p}<.000$), but not for Group or Age. While the mean efficiency score for the older students was higher than that for the younger students (see Table 3 and Figure 4), the difference was not significant.

Table 3

Mean Plan Route Efficiency Scores for First and Last Plans

(Study 2 Pretest)

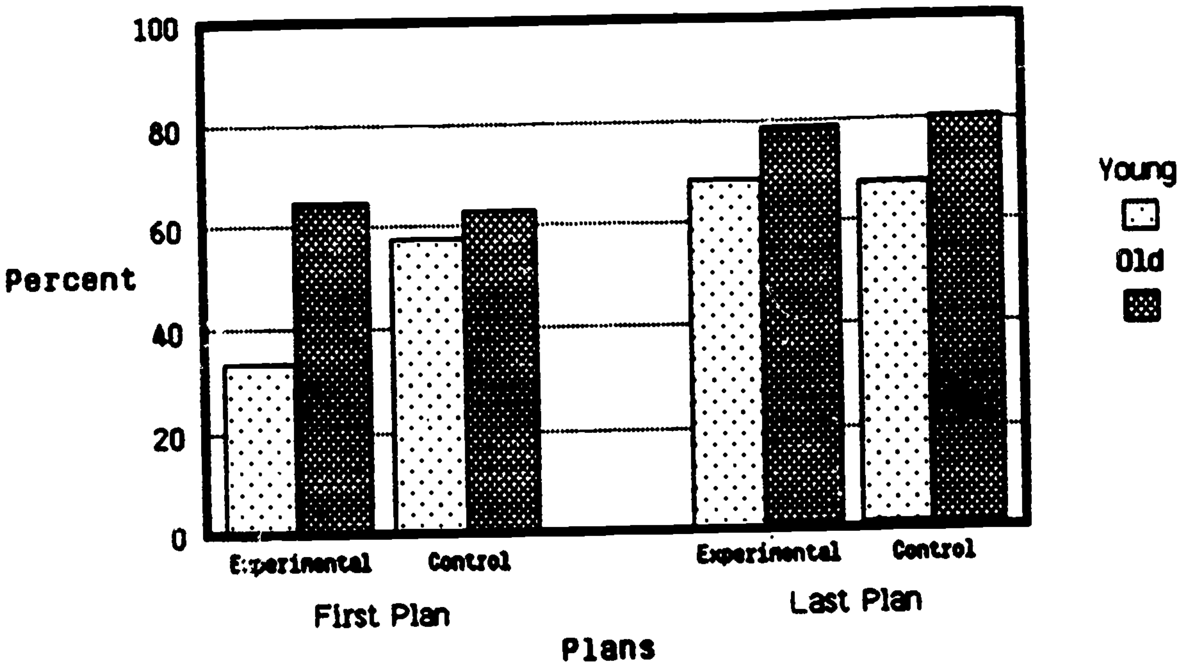
	First Plan		Last Plan	
	mean	sd	mean	sd
Logo Programmers				
Younger	32.6	(33.3)	68.1	(22.5)
Older	64.7	(29.0)	78.6	(18.9)
Nonprogrammers				
Younger	56.6	(36.1)	68.2	(31.3)
Older	62.7	(16.0)	80.0	(13.0)

Since there were no significant differences in planning scores between the experimental and control classes prior to the Logo treatment, subsequent analyses are confined to the second session.

The purpose of this study was to investigate how students who have been doing computer programming for a school year differ in their use of planning skills from students without programming experience. Given our chore-scheduling task, how might programming-related effects be expected to manifest themselves? We hypothesized that students with programming experience might differ from their nonprogramming peers in four major respects:

Figure 4

Plan Route Efficiency - Study Two Pretest



1. Programmers should be better planners overall, since programming provides a rich environment for learning the utility of planning. Therefore, lengths of plans for the programming students should be less than those for nonprogrammers (i.e., total time would be significantly less for programmers).

2. Programmers should make more and better use of the feedback available, since programming teaches the utility of debugging partially correct procedures. This means that programmers should ask more often to see a listing of their plans (review plan) and refer more often to the list of remaining chores (check list) than nonprogrammers. In addition, in the programming group, differences on these dimensions between students in the feedback and no-feedback conditions should be greater than in the nonprogramming group.

3. Programmers, relative to nonprogrammers, should spend more time early in their first plan thinking over alternative plans (i.e., significantly more pauses and longer mean thinking time in the first third of the first plan). On subsequent plans, their thinking time should become more evenly distributed across the plan as they concentrate on debugging different parts of it.

4. Programmers should seek to improve or debug their first plan through successive refinements in subsequent plans, rather than trying a different approach each time. This means that, relative to the nonprogrammers, the degree of similarity between successive plans for programmers should increase across plans (i.e., percent plan overlap should be greater than for nonprogrammers and should be greater between plans two and three than between plans one and two).

To evaluate the first hypothesis--that programmers would construct shorter overall plans--a repeated-measures analysis of variance was performed with total time as the dependent variable. Group (Logo, no-Logo), Condition (feedback, no-feedback), and Age (younger, older) were then entered as between-subject factors, and Plan (first, second, or third) as a within-subject factor. As expected, there was a strong main effect for Age ($p<.000$), indicating that older students produced better plans overall than the younger students. There was also a significant main effect for Plan ($p<.000$). Separate F values were calculated comparing first plan to second plan, first plan to third plan, and second plan to third plan. These F values were evaluated against an F that had been adjusted by the amount specified by the Newman-Keuls test (Keppel, 1973). This analysis indicated that first plans were significantly different from both second and third plans, but that the second and third plans did not differ significantly from each other.

No other main effects or interactions were significant at the .01 level. Means for the main subgroups are presented in Table 4 and graphed in Figure 5. As can be seen, all groups tended to develop better plans with each attempt. However, even the best group was still almost 200 seconds above the best achievable time of 900 seconds. (For comparison, a group of 16 adults with no programming experience had an average third plan time of 979 seconds with a standard deviation of 54 seconds on this same task.)

Table 4

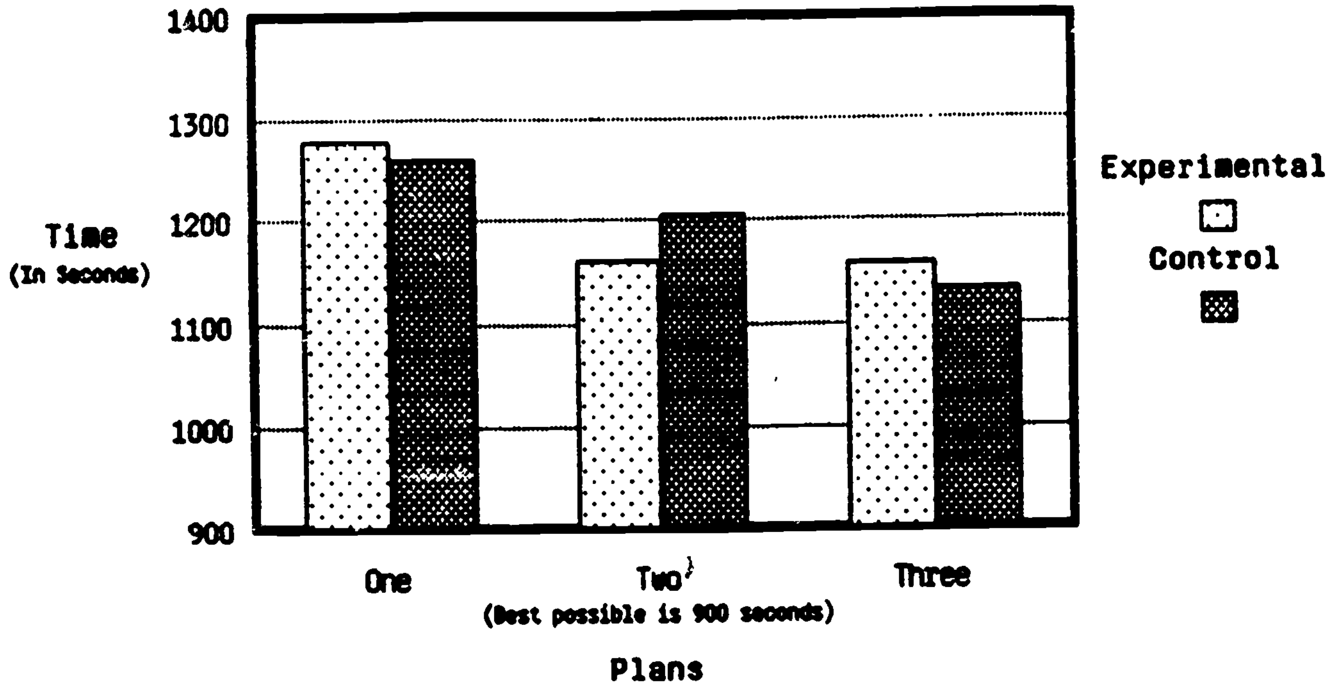
Total Execution Time of Plans (in Seconds)

	Plan 1		Plan 2		Plan 3	
	mean	sd	mean	sd	mean	sd
Logo Programmers						
Feedback:						
Younger	1376	289	1197	145	1179	114
Older	1190	150	1095	135	1113	119
No Feedback:						
Younger	1430	199	1282	180	1230	151
Older	1120	157	1075	141	1126	164
Nonprogrammers						
Feedback:						
Younger	1409	141	1235	143	1153	175
Older	1124	137	1167	172	1053	93
No Feedback:						
Younger	1228	257	1265	127	1149	143
Older	1283	202	1164	198	1183	194

To evaluate the second hypothesis--that programmers would make more use of the available feedback aids provided in the planning environment--separate analyses of variance were conducted for: (1) the number of times students checked the list of chores to be done; and (2) the number of times they asked to see a listing of their plans. Group, Condition, Age, and Plan were again the independent factors. Surprisingly, there were no significant main effects or interactions for either variable. The reason for the lack of effect

Figure 5

Average Execution Time of 3 Plans



appeared to be due to students' not using these features of the task environment. As can be seen in Tables 5 and 6, students in all groups rarely asked for the feedback that was available to them, even though there was a message on the screen at all times indicating its availability. In addition, students in the feedback condition tended not to spend much time watching their plan enacted on the screen, nor did they refer to the printed copy of earlier plans when creating a new plan.

Table 5

Number of Times Students Checked the List of Remaining Chores

	Plan 1		Plan 2		Plan 3	
	mean	sd	mean	sd	mean	sd
Logo Programmers						
Feedback:						
Younger	.75	.46	.25	.46	.13	.35
Older	.50	.53	.25	.71	.50	.76
No Feedback:						
Younger	.63	.74	.50	.76	.75	.71
Older	.13	.35	.25	.46	.13	.35
Nonprogrammers						
Feedback:						
Younger	.38	.52	.50	.76	.35	.13
Older	.38	.52	.25	.46	.52	.27
No Feedback:						
Younger	.50	.76	.13	.35	.00	.00
Older	.13	.35	.25	.46	.35	.13

To evaluate the third hypothesis--that programmers would be more reflective about their plans, and would spend more time thinking in the first third of the first plan than nonprogrammers--the total amount of thinking time per plan (see Table 7) and the average length of time between moves in the beginning, middle and end of each plan were analyzed (see Table 8 and Figure 6). In addition, the number of pauses in each plan was compared (see Table 9).

Table 6

Number of Times Students Asked to See Listing of Plan

	Plan 1		Plan 2		Plan 3	
	mean	sd	mean	sd	mean	sd
Logo Programmers						
Feedback:						
Younger	.88	.83	.63	.92	.25	.46
Older	.38	.74	.50	.76	.63	.92
No Feedback:						
Younger	.50	1.07	.25	.71	.38	.52
Older	.25	.46	.25	.46	.13	.35
Nonprogrammers						
Feedback:						
Younger	.38	.74	.13	.35	.25	.46
Older	.50	.76	.25	.46	.50	.76
No Feedback:						
Younger	.50	.53	.13	.35	.25	.46
Older	.38	.74	.50	.76	.50	.76

Table 7

Total Thinking Time (in Seconds) During Plan

	Plan 1		Plan 2		Plan 3	
	mean	sd	mean	sd	mean	sd
Logo Programmers						
Feedback:						
Younger	2088	1109	1276	892	1089	821
Older	2049	1076	1535	602	1305	431
No Feedback:						
Younger	1726	562	1142	310	902	175
Older	2041	738	1445	1101	994	410
Nonprogrammers						
Feedback:						
Younger	1808	714	1270	522	1029	348
Older	1248	526	977	542	1160	796
No Feedback:						
Younger	1824	315	1013	273	807	230
Older	1717	753	1151	614	1212	873

Table 8

Mean Thinking Time in Each Third of a Plan

		Plan 1		Plan 2		Plan 3	
		mean	sd	mean	sd	mean	sd
<u>Logo Programmers</u>							
Feedback:							
Younger	1st	6.27	3.94	5.40	6.37	4.95	4.95
	2nd	4.94	2.67	2.75	1.47	2.32	1.09
	3rd	3.90	2.03	2.49	1.25	1.96	1.10
Older	1st	6.89	4.76	4.85	2.02	4.55	1.89
	2nd	5.15	2.83	4.75	3.02	3.27	.83
	3rd	3.16	.96	2.34	.93	2.47	1.21
No Feedback:							
Younger	1st	5.48	2.07	3.87	1.83	3.08	1.63
	2nd	3.81	1.72	2.75	.96	1.83	.71
	3rd	2.90	1.15	2.23	.69	2.23	1.11
Older	1st	7.72	3.52	5.05	5.41	2.82	1.31
	2nd	4.72	1.54	4.29	3.09	3.08	2.20
	3rd	3.27	1.53	2.88	.84	2.13	1.01
<u>Nonprogrammers</u>							
Feedback:							
Younger	1st	5.83	2.23	4.23	2.95	3.42	1.61
	2nd	3.56	1.90	3.11	.84	2.21	1.53
	3rd	3.36	.84	2.49	1.00	2.77	1.09
Older	1st	4.40	1.44	3.19	1.59	4.27	2.90
	2nd	2.65	.92	2.38	1.06	3.06	2.12
	3rd	2.63	1.51	1.93	1.07	2.05	.69
No Feedback:							
Younger	1st	7.72	3.11	3.34	1.85	3.12	2.22
	2nd	3.40	1.77	2.34	.59	2.00	.66
	3rd	3.09	1.32	2.26	.32	1.64	.37
Older	1st	6.33	2.55	3.81	2.17	4.68	4.29
	2nd	3.82	2.77	2.99	2.06	2.30	1.36
	3rd	2.79	1.30	2.24	.75	2.88	1.95

Figure 6

Average Thinking Time in Beginning, Middle and End of each Plan

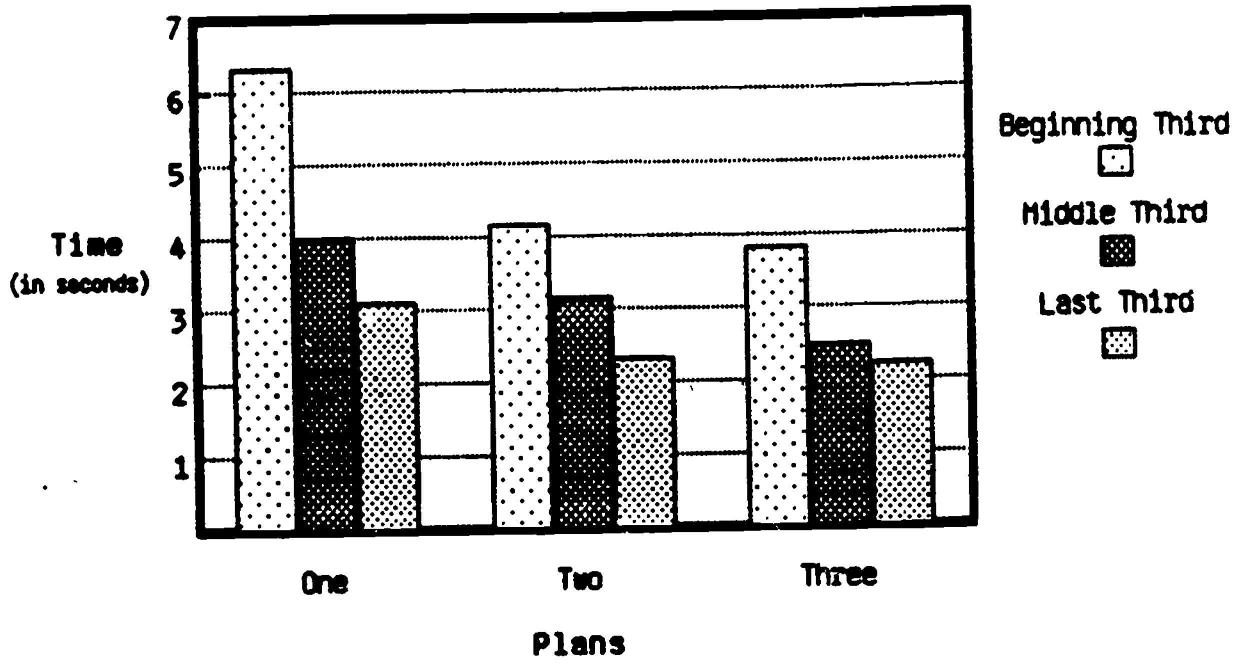


Table 9

Total Number of Pauses (Greater than 10 Seconds) During Plans

	Plan 1		Plan 2		Plan 3	
	mean	sd	mean	sd	mean	sd
<u>Logo Programmers</u>						
Feedback:						
Younger	4.38	3.93	1.75	1.91	1.38	1.77
Older	5.63	4.60	3.13	2.53	2.50	1.60
No Feedback:						
Younger	4.50	2.56	1.50	1.93	1.00	.53
Older	5.13	2.36	2.25	3.20	1.00	1.41
<u>Nonprogrammers</u>						
Feedback:						
Younger	4.50	3.63	1.63	1.06	1.50	1.41
Older	2.50	2.73	1.88	2.59	2.13	2.10
No Feedback:						
Younger	4.25	1.98	1.25	.89	.75	.46
Older	3.50	2.67	1.75	1.91	2.00	2.00

On total thinking time in plans, the only significant effect was a main effect for Plan ($\underline{p}<.000$). The Newman-Keuls test indicated that students thought significantly more during their first plan than in their second or third, but that the amount of time they spent thinking in their second and third plans did not differ significantly. Similarly, the only significant effect involving the number of 10-second or greater pauses in a plan (Table 9) was a main effect for Plan ($\underline{p}<.000$). Post hoc comparisons showed the same pattern: the first plan was different from the other two, but the second and third were not different from each other. When thinking time was broken down into thirds, corresponding to the average thinking time in the beginning, middle and end of the plan, in addition to the significant main effect for Plan ($\underline{p}<.000$), there was also a significant main effect for Position (first, second, or third part of plan; $\underline{p}<.000$), and a significant Plan by Position interaction ($\underline{p}<.000$). As shown in Figure 6, students put more thinking time into the beginning third of a plan than into the middle or end third. They also spent more time thinking on their first plan than on their second or third. Thus, while the pattern of thinking time for the programmers conformed to what we had hypothesized, it did not differ as predicted from the pattern for nonprogrammers.

To evaluate the fourth hypothesis, that programmers would have a greater tendency to debug plans by successive refinement rather than trying different approaches on each attempt, the amount of overlap from plan to plan (plan similarity) was analyzed. As can be seen in Table 10, the successive plans for all groups tended to overlap from plan to plan between 35% and 55%. A repeated-measures analysis of variance indicated that there were no significant effects for Group, Condition, or Plan. Thus, there was no evidence that the programmers were more likely to follow a model of plan debugging by successive refinement than nonprogrammers.

Table 10

Percent Overlap Between Successive Plans

	Plans 1 & 2		Plans 2 & 3	
	mean	sd	mean	sd
Logo Programmers				
Feedback:				
Younger	56	25	45	18
Older	42	23	42	14
No Feedback:				
Younger	54	10	55	15
Older	38	18	52	16
Nonprogrammers				
Feedback:				
Younger	36	22	37	25
Older	35	16	45	19
No Feedback:				
Younger	42	17	54	16
Older	56	21	43	26

In addition to asking whether the programmers produced plans more similar to each other than nonprogrammers, we were interested in whether there was any relationship between percent plan overlap (i.e., refinement strategy) and how good (fast) the plan was.

Therefore, percentage of plan overlap was correlated with the other main planning variables. As can be seen in Table 11, there was a small but significant relationship (_r_ = -.26) between degree of overlap from plan two to plan three and the execution time of plan three. However, this was not the case for the degree of overlap between plans one and two and the length of plan two (_r_ = .03). In general, it appeared that students who modified previous plans, leaving larger portions intact, did not develop appreciably better plans than students who varied their approaches from plan to plan.

Table 11

Correlation of Plan Similarity with Time for Robot to Execute Plan (N=64)

	1	2	3	4	5	6
1. Similarity Plans 1&2	--					
2. Similarity Plans 2&3	.33**	--				
3. Similarity Plans 1-3	.12	.42***	--			
4. Time of Plan 1	.11	.02	.12	--		
5. Time of Plan 2	.03	.03	.08	.16	--	
6. Time of Plan 3	-.07	-.26*	-.10	.28**	.32**	--

*$p<.05$
**$p<.01$
***$p<.001$

Conclusions

We examined the hypothesis that learning Logo programming contributes to students' planning skills by comparing the activities of groups of programming and nonprogramming students on several different planning tasks. In studies one and two, students who had spent a year programming did not differ on various developmental comparisons of the effectiveness of their plans and their processes of planning from same-age controls who had not learned to program. Results from study two are particularly striking in this respect, since the computerized planning task was designed to have a strong resemblance to programming, including feedback in different representational media (e.g., picture of plan in execution, list of moves in plan) that, because of their planning experience, programmers might have used to greater advantage. We made this effort to have the task resemble programming so that we could observe whether or not there was a transfer of planning skills, but the programming groups clearly did not use the cognitive abilities alleged to be developed in Logo in these tasks designed to tap them.

What do we conclude from these findings? There does not appear to be automatic improvement of planning skills from learning Logo programming. Why? We believe there are two major categories of possible explanations. First, there are objections to the tasks we have used and our resultant data. One argument says that these tasks do not tap planning skills. But the tasks have great surface validity, and the route efficiency measures in particular were developmentally sensitive, within and across sessions, and across age groups in general. Even the adults we tested in the experiment two task were not at ceiling performance. Thus, the developmental gap between actual performance and optimal performance could have been influenced by the greater development of planning abilities through programming. But whether or not a student programmed did not account for the variability we found in planning task performances. A second objection to our planning tasks is that they are not close enough to programming tasks for the transfer of planning skills from the programming domain to have been likely. But this is a moot point since, in both studies one and two, according to the Logo hypothesis, transfer of the concepts and practices of planning was expected to occur spontaneously, not because of the resemblance of the target task to the programming domain.

The second category of explanations concerns Logo programming, and here we may distinguish among three different kinds of arguments. First, there are problems with the Logo programming environment (not the instructional environment) as a vehicle for learning these generalizable cognitive skills. Second, the quality of learning about and developing such planning skills with the Logo discovery-learning pedagogy is insufficient for the development of generalizable planning skills. This is an objection to the "learning without curriculum" that Papert advocates for Logo programming. Third, perhaps the amount of time students spent in the Logo pedagogical environment was not sufficient for us to see the effects on planning of Logo programming experience. Children may need more Logo experience of the type they were getting in conjunction with the Logo discovery-learning educational philosophy. It should be noted in this context that, given the current limitations of computer resources and time in schools, and the rich array of the currently available uses of classroom microcomputers as tools for problem solving (e.g., as word processors, database management systems, electronic spreadsheets), it would take a great leap of faith to persist in fostering years of Logo discovery learning whose effects cannot be demonstrated.

On the basis of these two studies, we cannot tease apart these three alternatives. Although there may be problems with current implementations of Logo technology (e.g., Tinker, 1982), we do not see inherent limitations to Logo as a vehicle for developing planning and thinking skills. From our perspective, the great challenge is in our second argument, which we consider to be the most likely source of the lack of planning skill transfer in our studies. Learning how to plan well is not intrinsically guaranteed by the Logo programming environment; it must be supported by teachers who, tacitly or explicitly, know how to foster the development of planning skills through a judicious use of examples, student projects, and direct instruction. But the Logo instructional environment that Papert (1980) offers to educators is devoid of curriculum, and lacks an account of how the technology can be used as a tool to stimulate students' thinking about such powerful ideas as planning and problem decomposition. Teachers are told not to teach, but are not told what to substitute for teaching. Thinking skills curricula are beginning to appear, but teachers cannot be expected to create them spontaneously, any more than students can be expected to induce lessons about the power of planning methods from self-generated programming projects.

At a minimum, what we have demonstrated is that the strong claims for generalizable cognitive benefits from learning to program in Logo with the discovery-learning pedagogy need serious reexamination. In spite of the fact that our studies were conducted in relatively computer-rich classrooms--with one computer for each four students (as compared with the current ratio of one computer for every 183 students in elementary schools that have microcomputers [Becker, 1983])--we did not observe the predicted cognitive benefits of greater planning skills in students learning Logo programming.

Nonetheless, we applaud the widespread interest shown by educators (especially revealed through Logo use) in helping students to develop problem-solving skills of a generalizable nature, in contrast to a previous focus on "fact" learning that characterizes much of the programming and computer literacy curricula (Pea & Kurland, 1984). However, if we think that learning general problem-solving skills is important, these results indicate that we cannot expect this to happen spontaneously in the short space of a school year. More generally, there is no currently available programming language or computer environment that can, in itself, without instructional guidance, help students to develop these advanced thinking strategies, or make them aware of the broad range of problem domains to which they might be applied. At the current time, learning these skills is a deeply social affair, and it is by and large through interactions with skilled teachers that skill in problem solving and planning develops. The current range of programs to teach thinking skills attests to the interest in these goals as cognitive objectives of education. With limited resources, students may come to a deeper understanding of problem-solving skills from being taught such heuristics rather than inducing them in a spontaneous learning-discovery environment such as Logo.

Footnotes

[1]It is possible for the route efficiency score to be greater than 100 (a "perfect" score) because of score adjustments due to students' omitting chore acts and ending up with partial plans. Washing the paintbrushes was a chore act forgotten by many students, as was erasing the blackboards before washing them. Extensive forgetting of chores was rare, and experimental versus control groups did not differ in the number of omitted chores. To make the total distance of each plan comparable from one student to the next, an adjustment method was used to build up each partial plan to a "full" plan, i.e., one accomplishing every chore in the task. This conservative adjustment consisted of calculating for each plan for each individual the "median length of moves" (more meaningful than the mean, since interchore act distances were sometimes very small or very large). In order to derive "total plan distance," we added to the student's partial-plan distance the product of the number of omitted acts and their value for median move length.

[2]The fourteen plan features were nine different "chore act clusters," and five features involving "movables" (e.g., brushes, watercan, sponge). For the chore act clusters, improvements in plan structure occurred when:

1. Clusters 1 to 4. Each of the four tables with chairs was only visited once, at which time the table was washed and the chairs at that table were pushed in.

2. Clusters 5 to 7. During the only visit to the art table, five component acts were dealt with in a cluster: the table was washed, the trashpaper and paintbrushes were picked up, and the two nearby plants were watered (Cluster 7). Cluster 6 included any four of these acts, and Cluster 5 any three of them.

3. Clusters 8 to 9. Each of the two blackboards was only visited once, at which time it was erased and washed (Cluster 8 for one blackboard, Cluster 9 for the other).

For features of plans involving "movables," improvements in plan structure occurred when:

4. Cluster 10. Going to the sink, both instruments (sponge, watercan) that would be needed during a sweep around the room were picked up.

5. Clusters 11 and 12. The sink was not returned to until all three movable things (sponge, watercan, paintbrushes) were no

longer needed for other component chore acts (Cluster 12). Cluster 11 was returning any two of these three movables at once.

6. Clusters 13 and 14. Instruments at the sink (sponge, watercan) were picked up once rather than each time they were needed (e.g., getting the sponge to clean the blackboard, returning it, getting it to wash the tables). We designate getting the sponge only once as Cluster 13 and getting the watercan only once as Cluster 14, although neither is literally a cluster of acts.

References

Abelson, R. P., & diSessa, A. Turtle geometry. Cambridge, MA: MIT Press, 1981.

Anderson, J. R. Acquisition of cognitive skill. Psychological Review, 1982, 89, 369-406.

Atwood, M. E., & Jeffries, J. Studies in plan construction I: Analysis of an extended protocol (Technical Report SAI-80-028-DEN). Englewood, CO: Science Applications, 1980.

Bamberger, J., & Schon, D. A. Learning as reflective conversation with materials: Notes from work on progress (Working Paper No. 17). Cambridge, MA: Massachusetts Institute of Technology, Division for Study and Research in Education, December 1982.

Becker, H. J. School uses of microcomputers: Reports from a national survey. John Hopkins University, Center for Social Organization of Schools, October 1983, Issue No. 3.

Bereiter, C. Writing. In R. Tyler & S. White (Eds.), Testing, teaching, and learning. Washington, DC: The National Institute of Education, 1979.

Brener, R. An experimental investigation of memory span. Journal of Experimental Psychology, 1940, 26, 467-482.

Brooks, F. P., Jr. The mythical man-month: Essays on software engineering. Reading, MA: Addison-Wesley, 1982.

Brown, A. L., Bransford, J. D., Ferrara, R. A., & Campione, J. C. Learning, remembering and understanding. In J. H. Flavell & E. Markman (Eds.), Cognitive development (Vol. 3), of P. H. Mussen (Ed.), Handbook of child psychology (4th ed.). New York: Wiley, 1983.

Byte. Special issue on Logo, August 1982.

Case, R. Structures and strictures: Some functional limitations on the course of cognitive growth. Cognitive Psychology, 1974, 6, 544-573.

Case, R., & Globerson, T. Field independence and central computing space. Child Development, 1974, 45, 772-778.

Coburn, P., Kelman, P., Roberts, N., Snyder, T. F. F., Watt, D. H., & Weiner, C. Practical guide to computers in education. Reading, MA: Addison-Wesley, 1982.

diSessa, A. Phenomenology and the evolution of intuition. In D. Gentner & A. L. Stevens (Eds.), Mental models. Hillsdale, NJ: Erlbaum, 1983.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., & Solomon, C. Programming languages as a framework for teaching mathematics (Report No. 1899). Cambridge, MA: Bolt Beranek & Newman, 1969.

Feurzeig, W., Horwitz, P., & Nickerson, R. Microcomputers in education (Report No. 4798). Bolt Beranek & Newman: Cambridge, MA, 1981.

Flavell, J., & Draguns, J. A microgenetic approach to perception and thought. Psychological Bulletin, 1957, 54, 197-217.

Galanter, E. Kids and computers. New York: Putnam, 1983.

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

Goldin, S. E., & Hayes-Roth, B. Individual differences in planning processes. Rand Corporation Note N-1488-ONR, June 1980.

Goldstein, I., & Papert, S. Artificial intelligence, language, and the study of knowledge. Cognitive Science, 1979, 1, 84-123.

Hayes, J. R., & Simon, H. A. Psychological differences among problem isomorphs. In N. Castellan, Jr., D. Pisoni, & G. Potts (Eds.), Cognitive Theory (Vol. II). Hillsdale, NJ: Erlbaum, 1976.

Hayes-Roth, B. Estimation of time requirements during planning: The interactions between motivation and cognition. Rand Corporation Note N-1581-ONR, November 1980.

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

Heller, J. I., & Greeno, J. G. Information processing analyses of mathematical problem solving. In R. Tyler & S. White (Eds.), Testing, teaching, and learning. Washington, DC: The National Institute of Education, 1979.

Keppel, G. Design and analysis: A researcher's handbook. Englewood Cliffs, NJ: Prentice-Hall, 1973.

Klahr, D., & Robinson, M. Formal assessment of problem solving and planning processes in preschool children. Cognitive Psychology, 1981, 13, 113-147.

Kotarbinski, T. Praxiology: An introduction to the sciences of efficient action. Oxford: Pergamon Press, 1965.

Laboratory of Comparative Human Cognition. Culture and cognitive development. In W. Kessen (Ed.), History, theory and methods (Vol. I), of P. H. Mussen (Ed.), Handbook of child psychology (4th ed.). New York: Wiley, 1983.

Lyons, D. R. Individual differences in immediate serial recall: A matter of mnemonics? Cognitive Psychology, 1977, 9, 3-411.

Minsky, M. Form and content in computer science. Communications of the ACM, 1970, 17 (2), 197-215.

Newell, A., & Simon, H. A. Human problem solving. Englewood Cliffs, NJ: Prentice-Hall, 1972.

Nickerson, R. S. Computer programming as a vehicle for teaching thinking skills. Thinking: The Journal of Philosophy for Children, 1982, 4, 42-48.

Papert, S. Teaching children thinking. Programmed Learning and Educational Technology, 1972a, 9, 245-255..

Papert, S. Teaching children to be mathematicians versus teaching about mathematics. International Journal for Mathematical Education, Science and Technology, 1972(b), 3, 249-262.

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: Massachusetts Institute of Technology, Department of Artificial Intelligence, 1979.

Pea, R. D. What is planning development the development of? In D. Forbes & M. Greenberg (Eds.), New directions in child development: The development of planful behavior in children. San Francisco: Jossey-Bass, Fall 1982.

Pea, R. D., & Kurland, D. M. On the cognitive prerequisites of learning computer programming (Technical Report No. 18). New York: Center for Children and Technology, Bank Street College of Education, 1983.

Pea, R. D., & Kurland, D. M. On the cognitive effects of learning computer programming. New Ideas in Psychology, 1984, 2 (1), in press.

Pennington, N. Cognitive components of expertise in computer programming: A review of the literature (Technical Report No. 46). Ann Arbor: University of Michigan Center for Cognitive Science, July 1982.

Polya, G. How to solve it (2nd ed.). Garden City, NJ: Doubleday, 1957.

Rogoff, B., & Gardner, W. P. Developing cognitive skills in social interaction. In B. Rogoff & J. Lave (Eds.), Everyday cognition: Its development in social context. Cambridge, MA: Harvard University Press, 1983.

Ross, P., & Howe, J. Teaching mathematics through programming: Ten year on. In R. Lewis & D. Tagg (Eds.), Computers in education. Amsterdam: North-Holland, 1981.

Scribner, S., & Cole, M. The psychology of literacy. Cambridge, MA: Harvard University Press, 1981.

Simon, H. A., & Hayes, J. R. The understanding process: Problem isomorphs. Cognitive Psychology, 1976, 8, 165-190.

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.

Werner, H. Microgenesis and aphasia. Journal of Abnormal and Social Psychology, 1956, 52, 347-353.

Wertheimer, M. Productive thinking (Enlarged ed.). New York: Harper & Row, 1961.

APPENDIX

CODING CATEGORIES AND DEFINITIONS FOR PROCESS ANALYSIS

A. Decision Type Categories

The coding categories have been slightly modified from Hayes-Roth and Hayes-Roth (1979) and Goldin and Hayes-Roth (1980), but are comparable on most points. The type categories of analysis specify different conceptual categories of decisions made during the planning process. The first three categories of decisions choose plan features, the other two are more strategic in nature, determining features of the planning process.

1. Plan: represent specific actions the planner intends to take in the world (e.g., "go to wash the art table this way" while tracing out a path).

2. Plan abstraction: select desired attributes of potential plan decisions, noting kinds of actions that might be useful without specifying the actual actions (e.g., "go to closest chore next" or "organize plan around spatial clusters of chores").

3. World knowledge: assess data (e.g., of chore or instrument locations, distance, or time) concerning relationships in the task environment that might affect the planning process (e.g., "the hamster is next to the door" or "the chores are all in a circle").

4. Executive: determine allotment of cognitive resources during planning, such as what kinds of decisions to make first, or what part of the plan to develop next (e.g., "I'll decide what order to do the chores in before figuring out a path").

5. Metaplan: reflect planner's approach to the planning problem, methods they intend to apply to it, or establish criteria to be used for making up and evaluating prospective plans.

B. Abstractness "Level" (Within Type) Categories

For the "abstractness level" categories of analysis, decisions at each more specific or concrete level specify a more detailed plan than those at the higher level of abstraction. Levels for all the types but "metaplan" are hierarchically organized. Level stratification moves in the definition charts from abstract to concrete down the list:

1. PLAN TYPE

Level	Definition
1A: Outcome	Determine which chores will be accomplished when plan is executed (e.g., "I'll definitely do the hamster and the plants").
1B: Design	Determine specific spatiotemporal approach to planned activities (e.g., "I'll do the chores by going in a circle").
1C: Procedures	Determine specific sequences of gross actions (e.g., "I would do the hamster, and then get the sponge" <u>without noting path</u>).
1D: Operations	Determine specific sequences of minute actions (e.g., noting the details of the path for a sequence of gross actions in the plan).

2. PLAN-ABSTRACTION TYPE

Level	Definition
2A: Outcome (Intentions)	Determine which <u>kinds</u> of chores are desirable to accomplish when plan is executed (e.g., "Do all the important chores").
2B: Design (Scheme)	Determine <u>kinds</u> of desirable spatiotemporal organizations of planned activities to achieve outcomes (e.g., "I'll organize a plan around clusters of chores").
2C: Procedures (Strategy)	Determine characteristics of desirable <u>kinds</u> of sequencing of gross level individual chore acts (e.g., "I'll do the closest chore next").
2D: Operations (Tactic)	Determine characteristics of desirable kinds of sequencing of the specifics of individual chore acts (e.g., "I'll take the shortest route to the next chore").

3. WORLD-KNOWLEDGE TYPE

World knowledge type decisions suggest decisions at the corresponding plan abstraction level, or instantiate decisions at the corresponding plan level.

Level	Definition
3A: Outcome (Chores)	Note facts or values regarding specific chores to be accomplished (e.g., "feeding the hamster is the most important chore" or "washing blackboards takes a long time").
3B: Design (Layout)	Note facts of spatiotemporal organization of a group of planned activities (e.g., "there are a lot of things to do by the sink").
3C: Procedures (Neighbors or Instruments)	Note facts regarding the world of the chores relevant to ordering individual chore acts (e.g., "the closest chore to where I am now is watering Plant 1"; "Oh, I have to go get the sponge first").
3D: Operations (Routes or Chore Act Details)	Note facts that relate to the specifics of performing specific chore acts or travelling from one chore act to another (e.g., "through the benches is the shortest way to get to the blackboard" or "I can hold the watercan in my hand while I'm doing that chore").

4. EXECUTIVE TYPE

Level	Definition
4A: Priority	Establish principles for allocating cognitive resources during the entire planning process (e.g., "I'll decide what to do before deciding when to do things").
4B: Focus	Indicate what kind of decisions to make at a particular point in the planning process (e.g., "Now I'll figure out the shortest way to get over to the trashcan").
4C: Scheduling	Resolve any remaining conflicts between competing decisions that have been made, choosing one to execute next in the plan of action.

5. METAPLAN TYPE

Level	Definition
5A: Problem Definition	Define the planner's representation of the task and its goals, resources, and constraints.
5B: Problem-Solving Model	Define the general strategy the planner takes in making up a solution to the planning problem.
5C: Policies	Note a set of global constraints and desirable features for the developing plan.
5D: Evaluation Criteria	Define a set of dimensions against which tentative plans may be evaluated.