DOCUMENT RESUME

ED 275 700 TM 860 600

AUTHOR Magliaro, Susan; Burton, John K.
TITLE Adolescents' Chunking of Computer Programs.
PUB DATE Apr 86
NOTE 19p.; Paper presented at the Annual Meeting of the American Educational Research Association (67th, San Francisco, CA, April 16-20, 1986).
PUB TYPE Speeches/Conference Papers (150) -- Reports - Research/Technical (143)

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS Adolescents; Analysis of Variance; Cognitive Ability; *Computer Science Education; *Computer Software; *Pattern Recognition; *Programing; *Recall (Psychology); Secondary Education; *Serial Learning; Summer Programs
IDENTIFIERS BASIC Programing Language; Chunking; PASCAL Programing Language

ABSTRACT

To investigate what children learn during computer programming instruction, students attending a summer computer camp were asked to recall either single lines or chunks of computer programs from either coherent or scrambled programs. The 16 subjects, ages 12 to 17, were divided into three instructional groups: (1) beginners, who were taught to program in Applesoft BASIC; (2) intermediate, who were taught advanced concepts such as text files in Applesoft BASIC; and (3) advanced, who already had a background in BASIC and were taught PASCAL. The instruction involved programming syntax, debugging, planning, and use of a top-down programming structure. BASIC programs of 16 to 18 lines in length were arranged in a top-down structure or scrambled to separate lines which formed coherent procedural chunks. Students had two minutes to study the program and four minutes to write it down. Numbers of correct lines and chunks recalled were analyzed for each ability group and program version. All groups indicated greater recall of the coherent programs, especially the intermediate group. Increasing ability was associated with recall of the scrambled programs. Advanced programmers also commented that the scrambled programs did not make sense. (GDC)

Adolescents' Chunking of Computer Programs

Susan Magliaro and John K. Burton

Virginia Polytechnic Institute

and State University

Running Head: ADOLESCENTS' CHUNKING

Adolescents' Chunking of Computer Programs

In the last decade, much has been made of the cognitive benefits of computer programming for children (e.g., Papert, 1980). Students have been taught such programming languages as BASIC and LOGO with the goals of increasing their planning abilities, problem-solving skills, and metacognitive awareness of the problem-solving process itself. However, the research to date has produced contradictory results concerning the effects of learning to program on the development of children's complex cognitive skills (Pea & Kurland, 1984). At this point in time, it seems logical to focus research efforts on more basic questions. Specifically, what children do learn about the programming process and how do they organize that knowledge in memory?

Researchers of the psychology of computer programming has employed an information processing perspective in their attempts to uncover knowledge about the cognitive processes of programming (e.g., Brooks, 1977; Mayer, 1981). Specifically, problem solving theories have served as a theoretical framework for the investigation of the cognitive processes required to construct workable computer programs (e.g., Newell & Simon, 1972).

In focusing on computer programming, recent studies using adult subjects suggest that expert programmers have a "plan library" of basic computer programming "schemas," or recurrent functional chunks of programming code that are often used (Pea & Kurland, 1983). In an investigation of the knowledge

structures of programmers with varying levels of expertise, McKeithen et al. (1981) found that experts could recall more program lines and procedural chunks across trials than either intermediate or novice programmers. Further, experts were able to recall larger and more highly-organized chunks of a coherent program (using the ALGOL W programming language) than were intermediate or novice programmers. However, when students were given a scrambled version of the program, no differences were found across the groups. These results suggest that coherent programs were more meaningful for the experts, thus allowing them to apply what they knew about programming to the recall task.

The information derived from the recall protocols of novices indicated that these individuals focused on surface structures of the program. That is, they used mnemonic techniques based on orthography to enhance recall. The intermediates recalled small chunks of programming code, typically those expressed in natural language terms (i.e., IF-THEN-ELSE and FOR-STEP-WHILE-DO). Expert programmers demonstrated a great deal of clustering of the ALGOL W commands. These clusters were based on the function of the commands and the flow of control processes in the program. These results suggest that the experts' knowledge structures were richer and more complex than the structures of the intermediate or novice programmers.

In a related study, Adelson (1981) has found differences in memory capacity and the nature of the knowledge structures of expert and novice programmers. When asked to recall

programs in the Polymorphic Programming Language (PPL) code, experts were able to recall more of the program than novices. Further, the findings suggest that while both groups of programmers had conceptual categories for the elements of PPL, the expert programmers seemed to have a more complex conceptualization of the code. That is, the novice programmers' recalls were in the form of a syntax-based organization, while the expert programmers' recalls assumed a semantic, hierarchical organization based on principles of command functions.

Research on the changes in children's conceptual structures after instruction in a computer programming environment has just begun to emerge. For example, Pea & Kurland (1983) examined children's learning of programming syntax and comprehension of frequently-used subroutines. Their findings indicate that child programmers learn the basic syntax of the language and can construct simple, workable programs. However, they are not able to understand their own program's flow-of-control or more advanced concepts of recursion, conditionals, and variables.

In an effort to investigate what children do learn in a programming environment, the purpose of this study was to examine how students of varying programming abilities recall and organize programming concepts. Specifically, this study focused on the recall of advanced, intermediate, and beginning adolescents for coherent and scrambled computer programs. The specific research questions were:

1. What is the relationship between level of

programming expertise and the number of program lines and chunks recalled from coherent versus scrambled programs?

2. What is the relationship between level of programming expertise and the nature of line and chunk recall in coherent versus scrambled programs?

Design of the Study

Setting

The study was conducted during the second session of the Virginia Tech Computer Camp during the third and fourth weeks of July, 1985.

Treatment

On the first day of camp of the 12-day camp, all campers took a test on programming skills to facilitate placement in appropriate instructional groups: beginning, intermediate, and advanced. Beginning programmers were taught to program in Applesoft BASIC. Intermediate programmers were taught advanced concepts in Applesoft BASIC (e.g., text files). Advanced programmers had a background in BASIC and were taught PASCAL, a more structured, procedurally organized language than BASIC. At all levels, instruction involved the learning of programming syntax and debugging. In addition, the instruction emphasizes structured programming skills which included thoughtful planning before beginning on-line programming and the use of top-down programming structure. A typical daily schedule included approximately 5 hours of instruction, 4 of which were spent in planning programs or on-line programming.

Participants

From a pool of 50 campers, ages 10 to 18, 5 advanced programmers, 7 intermediate programmers, and 4 beginning programmers agreed to participate in the study. Four of the 16 participants were girls, 12 were boys. This ratio of girls to boys was approximately the same for the camp as a whole. Participants ranged in age from 12 to 17.

Materials

Two coherent and two scrambled versions of computer programs were constructed by the researchers and pilot tested during the first session of the camp (i.e., first two weeks of July). These programs reflected the concepts that were taught in the beginning classes and reinforced in the intermediate and advanced classes. All programs were 16 to 18 lines long and written in the Applesoft BASIC programming language. Coherent programs were arranged in a structured, top-down programming style, which inherently organizes the program into procedural chunks. Scrambled programs were constructed from programs consisting of the same concepts as the coherent programs with the program lines rearranged to separate program lines that formed coherent procedural chunks.

Data Collection

Data was collected on the tenth day of the two-week session. All participants were gathered in one classroom and given a packet containing the four programs, four pieces of loose-leaf paper, and a pencil. The presentation of the to-be-remembered programs was randomly ordered for each participant. The participants were instructed that they would

have two minutes to read the program and try to remember it as best as they could. At the end of two minutes, they were asked to close the program packet. Write on a clean piece of paper as much of the program as they could remember. The recall period lasted four minutes which appeared to be ample time for recall since all students completed their recall protocols, for scrambled and coherent programs, before the end of the four-minute time period. Between each trial the recall sheets were collected and the participants engaged in a brief distractor task.

Analysis of the Data

The recall sheets were scored for number of correct programming lines. A program line was considered correct if the command was spelled correctly and subsequent syntax allowed the gist of the command line to be executed. For example, a recalled line would be counted correct if the original program line read, PRINT "Hi, I'm glad to meet you.", and the participant wrote, PRINT "Hi, nice to meet you." Also if the participant changed the variable letters in a loop (e.g., FOR I = 1 to 5; to FOR S = 1 to 5), the program line was counted correct if all subsequent lines using the variable had assumed the correct variable assignment. The recall sheets were also scored for the number of procedural chunks recalled by the participants. Descriptive statistics were calculated for number of correct program lines and number of procedural chunks recalled. Two separate 3 x 2 x 2 split plot analyses of variance were conducted to assess group differences in line and chunk recall for each version of the coherent and scrambled

programs. Tukey tests were used to further investigate main effects within specific cells. Finally, protocols were qualitatively analyzed to examine the nature of the line and chunk recall.

Results and Discussion

Table 1 reports the mean number of program lines and procedural chunks recalled across program conditions and groups. The raw data indicated that there was a difference in line and chunk recall according to program type (coherent versus scrambled). All three groups of programmers recalled more program lines and chunks from the coherent programs than from the scrambled programs. From the graphs of the raw data (see Figures 1 and 2), this difference was found to be especially apparent for the intermediate programmers. This finding may be due to the curricular emphasis at the computer camp, as well as the programmers' experience level. The advanced group, although familiar with the BASIC language, had been programming in PASCAL during their time at computer camp. These campers may not have been as able as the intermediate group to spontaneously apply schemata or templates for coherent models of BASIC programs. While both the intermediate and beginning programmers had been programming in BASIC, the intermediate group, with their increased level of experience and greater knowledge about what makes programs work, might have been more sensitive to the programs' level of coherence.

Insert Table 1 and Figures 1 and 2 about here

The results of the two analyses of variance confirmed these findings. There was a significant main effect for program version (coherent versus scrambled), for both line recall [$F(1,13) = 7.22$, $p<.017$] and chunk recall [$F(1,13) = 19.477$, $p<.001$]. While not reaching an acceptable level of significance (i.e., $p<.05$) in either the line [$F(2,13) = .497$, p=NS] or chunk [$F(2,13) = 3.350$, $p<.066$] recall analysis, an interaction between level of experience and program type can be seen in the graphs of the mean scores (see Figures 1 and 2). Contrary to the findings of Adelson (1981) and McKeithen et al. (1981), the advanced programmers did not recall the most program lines or chunks in the coherent condition. When compared with the advanced or beginning programmers, the intermediate programmers recalled more lines and chunks in the coherent condition, but not in the scrambled program condition. In the scrambled condition, the advanced programmers recalled more than the intermediates, who, in turn, recalled more than the beginners.

This interaction may be explained in terms of the development of a specific versus general knowledge of programming. That is, the primary and, for the most part, sole programming language of the intermediate programmers was the BASIC language. During their ten days at camp, their expressed tasks had been to construct coherent programs that were procedurally organized, resulting in a specific knowledge of a programming language. The advanced group, on the other hand, had a working knowledge of BASIC and were expanding their knowledge of programming to another environment -- PASCAL.

Thus, the advanced programmers were developing a broader, more general knowledge of programming. As a result, the advanced programmers were able to use this general knowledge to organize procedural chunks in the scrambled, as well as the coherent, programs, while the intermediate programmers had to rely on their specific knowledge base of procedurally-oriented BASIC programs.

Unlike the findings of McKeithen et al. (1981), the beginning programmers showed significant differences in both the number of lines ($p<.05$) and chunks ($p<.01$) recalled in the coherent versus scrambled conditions. While the chunking findings may be suspect due to the narrow range of the raw scores (i.e., the beginners recalled only two procedural chunks from the coherent programs, and none from the scrambled programs), it seemed that, given the mean difference in line recall, they had developed a level of expertise that could differentiate coherent from scrambled programs. While this knowledge was not sufficient to enable the beginning programmers to recall as many lines or chunks as the more experienced programmers, they were able to demonstrate some degree of knowledge of the domain.

Closer examination of the nature of the program line recall revealed two distinct patterns. First, the beginning programmers tended to group like commands together from the coherent, as well as the scrambled, programs. For example, one beginning programmer wrote all of the lines that called subroutines together (i.e., GOSUB 70, GOSUB 70, GOSUB 200). Another grouped all of the code that controlled the colors of

the programmed graphic (i.e., COLOR=1, COLOR=2, COLOR=0). Thus, while not forming chunks related to procedures, the beginners did organize their recall according to syntactic features. The advanced programmers, on the other hand, attempted to recall the scrambled, as well as the coherent, programs in a procedural order. That is, they seemed to try to make "sense" out of the scrambled programs by reordering some of the lines to produce workable procedural chunks.

The second response pattern paralleled the serial position effect reported by Rundus (1971) in his research on verbal rehearsal and word list recall. His findings indicated that the probability of recalling a word depended upon its position in a list. Typically, words appearing in the beginning or at the end of the list were recalled more often than words in the middle of the list. Words at the beginning of the list were rehearsed more often than the other words, giving them a higher probability of being retrieved from long-term memory. Words at the end of the lists were most recently seen, thus having them accessible in short-term memory. This same serial position effect was seen in beginning programmers' recall of coherent programs and all programmers' recall of scrambled programs. These patterns of recall support the notion that the beginning programmers relied on a rote recall based on syntactic or physical features for coherent and scrambled programs, while the more experienced programmers reverted to this method only when they could not extract some meaning from the programs.

These findings suggest that as adolescents gain expertise in programming, they develop specific schemata for often-used

chunks of programming procedures. Even the beginning programmers, who had only 10 days of programming, demonstrated a rudimentary knowledge of the BASIC language and procedures. The findings from the intermediate group indicated that their proficiency as programmers was advanced enough to be successful when the program made sense, but fragile enough to falter when the program was incoherent. The advanced programmers seemed to be able to take a program, either coherent or scrambled, and order the lines into meaningful procedural chunks that would allow the program to run successfully. Similar results were reported by McKeithen et al. (1981) with adult programmers and Egan and Schwartz (1979) with electronics experts. In the McKeithen et al. study, the adult expert programmers added specific lines to the scrambled program in order to produce coherent nested loops and output sequences. In the Egan and Schwartz study, the electronics experts attempted to recall symbols systematically when the symbols of the electronics diagram were inappropriately positioned.

Differences between advanced and intermediate programmers also became evident in their comments about the recall task. Only advanced programmers remarked about the incoherence of the scrambled programs and stated that those programs were more difficult to remember because "they didn't make any sense." Thus, the difference between intermediate and advanced programmers was seen in their desire to make the program "make sense" as well as in their confidence in declaring that something was wrong with the scrambled program.

Conclusions

The results of this preliminary investigation indicate that adolescents do learn and organize meaningful knowledge about computer programming much the same way that adults do. That is, those adolescents with more experience and practice develop a level of expertise that allows them to recall programs in terms of semantically related procedures rather than syntactic or physical features. In terms of this study, even those participants who were not programming in BASIC on a day-to-day basis were still able to recall their prior experience with BASIC and the procedures that produce certain programmed results.

While the generalizability of this study is restricted to this sample and setting, it is encouraging to see evidence that adolescents can acquire some levels of expertise in the area of programming. Continued study in this area will be able to reveal individuals' conceptual organization of this programming knowledge. Further investigation into the types of instructional environments and individual differences that are related to the acquisition of programming knowledge is also warranted. By assessing what children are learning in programming environments, we will be better able to ascertain the cognitive benefits of learning to program.

Acknowledgements

The authors wish to thank Norman Dodl, Director of the Virginia Tech Computer Camp, the staff members who assisted in the data collection, and the campers who graciously participated.

References

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. Memory and Cognition, 9, 422-433.

Brooks, R. E. (1977). Toward a theory of the cognitive processes in computer programming. International Journal of Man-Machine Studies, 9, 737-751.

Egan, D. E., & Schwartz, B. J. (1979). Chunking in recall of symbolic drawings. Memory and Cognition, 7, 149-158.

Mayer, R. E. (1981). The psychology of how novices learn computer programming. Computing Surveys, 13, 121-141.

McKeithen, K. B., Reitman, J. S., Reuter, H. H., & Hirtle, S. C. (1981). Knowledge organization and skill differences in computer programmers. Cognitive Psychology, 13, 307-325.

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

Papert, S. (1980). Mindstorms: Children, computers, and powerful ideas. New York: Basic Books.

Pea, R. D., & Kurland, D. M. (1984). On the cognitive effects of learning computer programming: A critical look (Report No. 9). New York: Bank Street College of Education.

Pea, R. D., & Kurland, D. M. (1983). On the cognitive prerequisites of learning computer programming

(Technical Report No. 18). New York: Bank Street College of Education.

Rundus, D. (1971). Analysis of rehearsal processes in free recall. Journal of Experimental Psychology, 89, 63-77.

Figure 1. Mean number of lines recalled across groups and conditions.

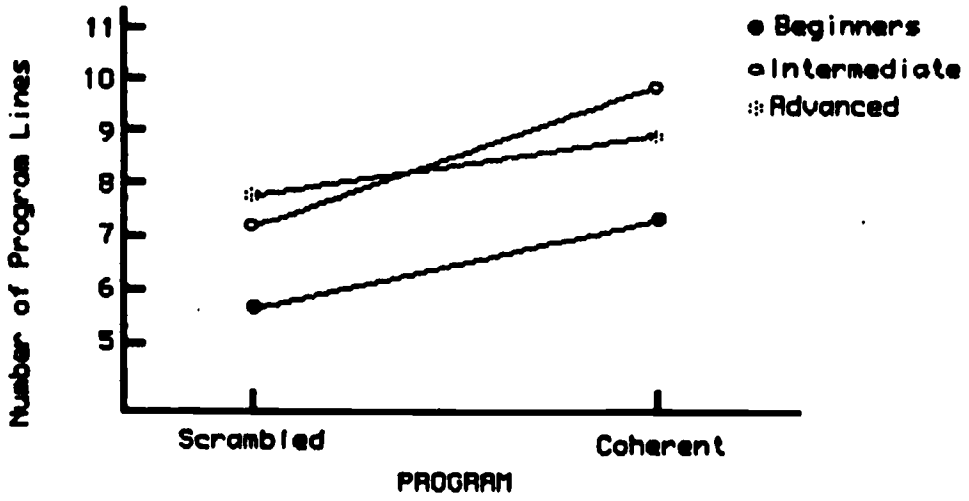

Figure 2. Mean number of chunks recalled across groups and and conditions.

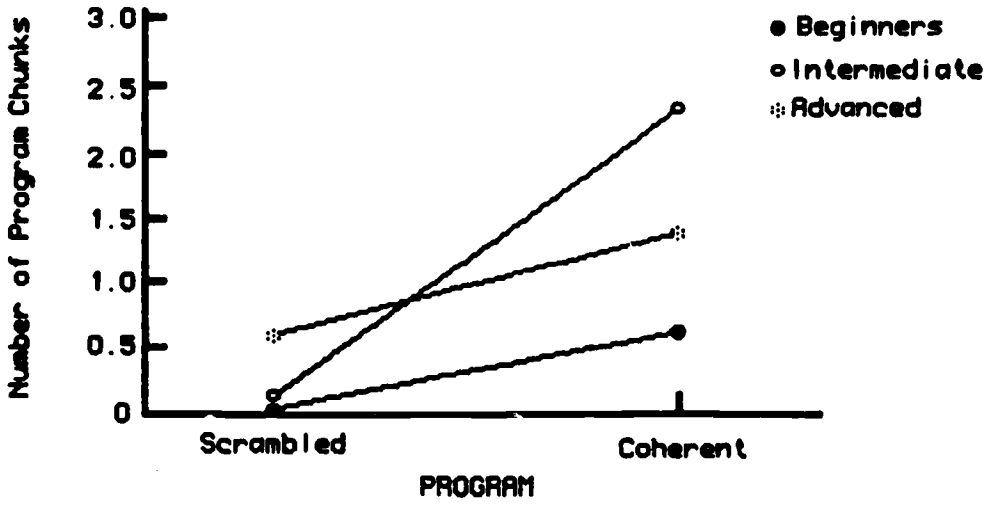

Table 1. Mean number of program lines and procedural chunks recalled across conditions and groups.

	Beginning		Intermediate		Advanced	
	Lines	Chunks	Lines	Chunks	Lines	Chunks
Coherent	7.38	0.63	9.93	2.36	8.90	1.40
(SD)	2.20	0.74	4.25	1.78	3.78	1.43
Scrambled	5.75	0.00	7.21	0.14	7.80	0.60
(SD)	1.28	0.00	2.42	0.36	1.93	0.84