Effects of Providing Graphic Models on Problem Solving with Logo

Gordon J. Mitchell

A Thesis

in

The Department

of

Education

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Arts at
Concordia University
Montréal, Québec, Canada

March 1985

ABSTRACT

Effects of Providing Graphic Models on Problem Solving with Logo

Gordon J. Mitchell

Research has established the facilitative effects of including concrete and graphic models as illustrative aids and advance organizers within instructional strategies directed toward the teaching of abstract or technical information. This thesis hypothesized that providing a model as an advance organizer using the relatively untapped graphic capabilities of the microcomputer would have a facilitative effect on comprehension and transfer of learning to novel problem solving situations.

Treatment groups received either an advance organizer graphic model or a post organizer graphic model, in addition to a traditional CAI lesson. As hypothesized, group means for the model groups were higher than for a control group on an immediate posttest, and significantly higher for the group that received the graphic model as an advance organizer. However, this difference became non-significant on a delayed posttest although the mean for the advance organizer group remained higher. It was concluded that the presentation of graphic models via the microcomputer can have a facilitative effect on learning, and that further research should address the possibility of providing review and practice strategies within CAL environments.

Acknowledgements

I would like to express my great appreciation to my thesis advisor, Prof. Jesus Vazquez-Abad, without whose patience and assistance I would never have been able to complete this thesis.

I would like to thank Laura Winer and Tom Wilson of the Department of Education, Concordia University and Frank Green of the Department of Education, McGill University, for making available the students who participated in this study.

I would also like to express my appreciation to the faculty and staff of the Department of Education, Concordia University, who in creating a relaxed, open environment, have made it considerably easier for students to pursue and complete their own research endeavours. Without this "unspoken" support, the completion of this thesis would have been a great deal more difficult.

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

CHAPTER I

Introduction

The development and application of computer technology has proceeded at an ever-increasing rate. More and more, computers are being used by people who possess little or no previous computer or programming experience as tools in a vast range of activities and disciplines. This has prompted demand for instruction in both computer applications and programming. In terms of computer programming, little is currently known about either the relevant skills involved or about how to teach them. There is an urgent need for research in this area (Coombs and Alty, 1981), both in terms of providing guidelines towards practical applications and in furnishing directions for further research.

There are innumerable research questions which need to be addressed with respect to the teaching of novice programmers (Du Boulay and O'Shea, 1981). What are the major difficulties encountered by novices in learning to program? What features make a language suitable or unsuitable as a first programming language? What are the skills to be imparted and how best can this be done in terms of teaching strategies? The review of literature presents current research addressing these questions.

The focus of this thesis involves the determination of effective instructional strategies for teaching abstract computing concepts to novice programmers. Novice programmers have been defined as "users who have had little or no previous experience with computers, who do not intend to become professional programmers, and who thus lack specific knowledge of computer programming" (Mayer, 1981). Novices are therefore seen as people who either want or need to understand and use computers as tools in applications not directly related to that of computer science. The numbers of such people are growing at an alarming rate, but because they come from a multitude of professions and disciplines, they do not form a cohesive group and consequently the difficulties they encounter with respect to computers and programming have not been adequately addressed. This thesis aspired to provide concrete guidelines towards making computers and computer programming more meaningful to novice programmers.

Card, Moran, and Newell (1983) present an interesting developmental theory of cognitive behaviour that raises some pertinent questions about the relationship between programming and problem solving within an educational environment. Situating their theory within that of contempory information-processing psychology, and especially in terms of cognitive skill acquisition (Anderson, 1980, 1981), they contend that all cognitive behaviour is located and progresses along a continuum from problem solving to cognitive skill. Probelm solving behaviour, then, will with practice become cognitive skill, and cognitive skills in turn start out in problem solving behaviour. The theory of Card, Moran, and Newell (1983) is described in the next chapter. The extension of their theory into practical application helps to justify the association between programming and problem solving within an educational environment. In terms of this thesis, it provides some justification for the use of problem solving as a performance measure for a population of novice programmers.

Problem Statement

As the introduction noted, there are many research questions which need to be addressed with respect to the teaching of novice programmers. The author believes that the question of effective instructional strategies is particularly germane to the field of educational technology. Is one strategy more effective than another in teaching a particular class of material? Is there one strategy that might maximize the teaching of basic programming concepts? Is there a significant interaction between individual learner aptitudes and instructional strategy? The answers to these questions will have to be furnished by future research.

This thesis has addressed the problem of how abstract programming operations such as word and list manipulations can be made more concrete and meaningful to novice programmers. Specifically, it investigated and applied several instructional strategies to the teaching of a subset of the programming language Logo, a high-level interactive language developed by Feurzeig et al (1969) as an aid to the teaching of mathematical concepts. The subset, which involves the word and list operations of Logo, is generally more abstract and difficult for the student to visualize than the better-known turtlegraphic subset popularized through the work of Seymour Papert (1980) among others. These abstract operations represent a difficult area for traditional instruction to present in a manner which is sufficiently concrete and comprehensible enough to allow students to develop a clear understanding of the hidden processes involved.

Logo word and list handling involves operations on words, list of words, and lists of lists, including operations that analyze inputs by breaking them down or combine inputs to form new words or lists. For example, the operation BUTFIRST takes a word or list as input and outputs the word or list minus the first character, word, or list of words. The operation WORD concatenates its input into one word. Similarly, the operation FPUT takes a word and list as input and outputs a new list formed by putting the word at the beginning of the list. Other operations search through a word or a list for a specific character or word, or test for equality between words or lists against other words or lists.

Students are typically asked to solve problems involving the manipulation of words and lists using these operations either singularly or in combination. For example, the student might be presented with a sentence and asked to write a procedure (a Logo program) that outputs every second word, or one that searches through the sentence for specific punctuation. Other problems might involve the use of recursion and variables in asking the student to write procedures to analyze input from the terminal or to design interactive CAI sequences.

The execution path of the processes used in such problems is not immediately transparent, and the novice programmer with no previous programming experience might not understand how or why the Logo procedures operate. For example, a student might not understand that a procedure that searches through a list for a specific word with a recursive BUTFIRST construction abbreviates the list by one word with each recursive call until it finds the specific word. Similarly, the operation of a procedure that reverses a word or list is not transparent in terms of the list-searching and recursive constructions involved.

When operations are combined within larger procedures (eg. PRINT FIRST BUTFIRST BUTFIRST (input)), the power of Logo as a word and list processing language can be more fully realized. Using Logo operations and commands in combination, the programmer is able to target specific elements of list, words, or characters for examination and manipulation. Unfortunately, however, the novice programmer might not understand how Logo operates upon combined program statements. For example, when two or more operations appear on a line, Logo executes the operations not in the left to right order expected by most students, but from right to left. This lack of understanding will certainly limit the programmer's ability to use the language to its fullest potential.

Logo is currently taught as a component of the CAL Program in the Education Department, Concordia University, Montreal. The author has had the opportunity to assist in the Concordia CAL laboratory during the fall and winter semesters, 1983/84, during which time he assisted graduate students in the learning of the command system and syntax of Logo. A questionnaire circulated at the conclusion of the fall semester, 1983, revealed a substantial dissatisfaction with both the documentation and the educational resources available to the students learning Logo.

Specifically, over 58% of the students strongly disagreed that the available Logo documentation and reference materials had been well designed, whereas only 8% of the total strongly agreed. All of those who did agree had either some or an extensive amount of computer and mathematical experience. Similarly, only 16% strongly agreed that the documentation had facilitated their understanding of Logo, whereas 84% disagreed. Similar opinions were expressed relating to the difficulty students encountered in learning to use the Logo command and file systems. On the other hand, the editing system posed no problems for almost 70% of the students. A total of 33% strongly agreed that they had had difficulty solving Logo problems on their own, whereas 25% felt that they had had little difficulty. In addition, over 69% of the students felt that at the end of the semester they were not able to use Logo word and list operations. Finally, over 72% of the students strongly agreed that a general introduction and overview to Logo would have been of assistance.

In summary, the questionnaire indicated that a majority of students felt that the available documentation and reference materials had neither been well designed nor helpful towards their understanding of Logo. Students with little or no mathematical or programming background had considerable difficulty learning to use the Logo command and file system. A majority felt that a more structured introduction would have been advantageous in terms of providing an overview to facilitate the solving of problems.

The author's personal experience in the Concordia CAL laboratory has indicated that while students at this level have little difficulty with the graphic subset of Logo, it is the word and list operations which cause the greatest difficulties. It has already been stated that over 69% of students questionned did not feel that they could apply these operations towards the solving of problems. Of the students who did feel comfortable with them, 82% had had previous mathematical and programming experience. It is believed, therefore, that instruction at the university level directed towards students with little or no mathematical or programming experience, should focus upon the non-graphic aspects of Logo.

This thesis attempted to define effective instructional strategies for teaching such abstract programming concepts as Logo word and list operations to a population of novice computer programmers. These strategies involved the development of an analogical model of Logo word and list processing. It was hoped that this would facilitate the comprehension and retention of the language by concretizing abstract concepts and processes through the use of familiar analogies, presented graphically via the microcomputer either before or after more traditional instruction. Such models have been variously labeled as virtual machines (Howe and Ross, 1981), notational machines (Du Boulay et al, 1981), conceptual windows (Sime and Fitter, 1978), and advance organizers (Mayer, 1975, 1976, 1981). A review of research addressing these areas, in addition to those of the difficulties encountered by novice programmers, programming as problem solving, and graphics and text in CAL follows. The author believes that the definition of effective instructional strategies incorporating analogical models would partially fill the gap between current educational methodology and one that will maximize learning for the individual student.

CHAPTER II

Review of the Literature

This review focuses on research investigating how novices learn programming concepts and use them as an aid in problem solving activity. Difficulties encountered by novices in learning to program are described, in addition to specific instructional strategies aspiring to mitigate these difficulties by making the processes involved in programming operations more concrete and meaningful. Such strategies include the use of advance organizers and concrete models to simulate and externalize operations that would otherwise remain hidden from the student. These strategies are considered to be highly effective by their proponents, and research supportative of this claim will be presented.

Teaching Novices Programming

A fair body of research has already investigated the difficulties encountered by novices in planning, coding, and debugging computer programs.

In terms of planning, Miller (1974) studied novices' abilities to construct algorithms of limited sorting problems and found that problems involving disjunctive decisions (one attribute OR another) resulted in greater logical error than problems involving conjunctive decisions (one attribute AND another). In a later study, Miller (1975) found that novices tended to produce incomplete algorithms in terms of not specifying what to do when a set of conditions was not satisfied. Weyer and Cannara (1975) reported that children demonstrated a misunderstanding of the concept of general algorithm in believing that the flowchart they had designed represented the actions the computer would take as a result of a specific set of inputs and not for a more general class of possible inputs. Children were instructed to complete a partially developed flowchart representing a chocolate-dispensing machine. Some misinterpreted the flowchart, believing it represented the action that the machine would take as a result of a single set of input coins rather than generalizing it as an algorithm to deal with all possible sets of coins.

With respect to the problems novices encounter in the coding process, Young (1974) has developed a classification of novices' coding errors, including syntactic, semantic, logical and clerical errors. Du Boulay and O'Shea (1981) add a fifth category, that of stylistic errors, from Kernghan and Plauger (1974).

Syntactic errors are improperly written expressions which the computer cannot interpret (eg. omitting a BEGIN or END statement in PASCAL). Semantic errors are instructions that, though semantically correct, instruct the computer to carry out impossible or contradictory actions (eg. reading a closed file). Logical errors are errors of poor planning or incorrect mapping from problem to program. The program doesn't do what it was intended to do. Clerical errors include mistyped characters, and are due to carelessness in the coding process. Finally, stylistic errors are errors of programming style that make a program difficult to read or interpret.

Young (1974) compared expert and novice programmers and concluded that novices made far more semantic errors than the experts, who made syntactic, semantic and logical errors with about equal frequency. Du Boulay and O'Shea suggest that this demonstrates that the novices were still uncertain about the properties of the "notional machine" defined by the specific language and its implementation. The concept of "notional machine", described by Du Boulay et al (1980), constitutes an internal model of the computer which is both concrete and precise enough to allow the learner to apply analogies with success. A more comprehensive discussion will be taken up later in this review.

Du Boulay and O'Shea (1981) contend that novices write poorly structured programs, lacking in style and comprehensibility, because of a combination of semantic, logical, and stylistic problems. Semantic problems they claim are due to a lack of experience with the virtual machine on which the program is to be run. Logical errors are attributed to a lack of familiarity with algorithms and flow of control. Finally, stylistic problems result from an unfamiliarity with stylistic rules and program techniques that enhance reliability.

Research into program debugging strategies shows not surprisingly that experienced programmers are more competent than novices (Gould, 1975). Young (1974) found that novices, unlike experienced programmers, had great difficulty with both semantic and logical bugs. Professionals had little trouble with semantic errors and spent most of their debugging time dealing with logical errors.

Several researchers have studied the manner in which programming languages are learned and internalized.

Hoc (1977) studied how programming languages are progressively internalized in what he terms a "Systeme de Representation et de Traitement" (SRT). He characterizes the term "device" as relating the conduct of a subject to his environment; as an objective interface performing according to defined rules a number of actions which in some manner alter the environment. He states, "As soon as a subject is able to operate a device mentally and predict the outcomes of his actions, even if his predictions are incorrect, we say that he has constructed a representation with the aid of which he can make calculations" (Hoc, 1977, p. 89).

Instruction, according to Hoc, consists in having the student construct the appropriate SRT's to represent the device language he is learning. He concludes that there are several steps involved in the interiorization of a language which lead the subject from a word-by-word translation of procedures from foreign SRT's into the object language, towards the using of SRT's that are an accurate reflection of the learned language itself.

Coombs et al (1982) conducted several studies which correlated learners' cognitive style with programming performance. Specifically, they studied the relationships between learning style as defined by Pask (1976) and two classes of learning activity considered necessary for programming competence:

1) The learning of individual language structure;
2) The learning to assemble these structures in a particular order to achieve specified objectives.

Utilizing Pask's operation/comprehension learning strategy continuum, they tested the ability of introductory FORTRAN students to both learn individual language statements and to assemble these structures into an operating program. Their conclusions bear repeating here:

1) It is possible to define at least two different learning styles in a population of novice computer users.
2) Students exercising one of the styles, operation learning, are more successful at assembling language structures into an effective algorithm.
3) The successful learning style is characterized by close attention to detail and a preference for procedural representation.
4) Success in the correct identification of individual language structures is independent of learning style.

(Coombs et al, 1982, p 466)

Operation learners, they found, tended to develop the macrostructures of a language more so than comprehension learners who did not develop these higher-level structures because they tended not to integrate and evaluate language structures after exposure to them. Operation learners developed by interacting with the computer their knowledge of the functional relations between low-level language structures, leading to a knowledge of macrostructure. They tended to work from inside the language, paying close attention to the procedural representation of logical relations between low-level language structures. Comprehension learners tended to rely on external factors and represented this knowledge in descriptive rather than procedural terms.

Coombs et al (1982) conclude that rather than forcing comprehension learners into an operational mold which Pask (1978) determined was not an effective strategy, it would be more advantageous to leave them work within their natural style, and to provide them with accurate conceptual and supporting information. The recommendation of Coombs et al (1982) of providing accurate conceptual information could perhaps be realized through the development of appropriate conceptual models. These models could assist in developing in those students who lacked them the faciliative language macrostructures possessed by operation learners.

Cannara (1976) has reported on the difficulties children encounter in learning Logo. He attributes much of this difficulty to their misunderstanding of the notional machine noted previously. Two of the most common misunderstandings on the part of the children were:

1) Misunderstandings of linguistic/computational context in terms of storage/passing of information within their programs and their interactions with the interpreter;
2) Ill-defined intents in terms of fuzzy program specifications or wishful thinking about how flexible the interpreter would be in executing poorly written programs.

Some children tried to run programs while the programs were being edited or vice versa. They also had problems with the storage and retreival facilities, in addition to misunderstandings concerning name/value and recursion/iteration distinctions. Some students believed that procedure names had to describe their actions in order for them to work, or that variable names were computationally related to their values.

Austen (1976) and Statz (1973) have done related work with adults, and suggest that both adults and children make similar types of mistakes when learning Logo. Statz (1973) reports that undergraduate students had difficulty in terminating a recursive procedure with a stop rule, in result-passing between sub-procedures and in the understanding of conditionals. In comparing adults with children in terms of programming performance, he noted that while both groups tended to make similar mistakes, the adults were better able to recover from them. Austin (1976) reports that the student teachers he instructed had difficulty with editing, filing, sub-procedurization, and syntax.

Du Boulay (1978) analyzed 2,400 error messages from 19,000 Logo commands issued by 15 student teachers. He found that calls to undefined procedures accounted for almost 30% of the total number and insufficient arguments for 16% of the total. Other errors included misused variables, the Logo turtle out of bounds, the wrong type of argument, and syntactical inconsistencies.

The learning of a programming language is thus seen variously as learning to effectively plan, code, and debug computer programs (Du Boulay and O'Shea, 1981; Miller, 1974, 1975; Young, 1974), as the internalization of language structures (Hoc, 1977), and similarly, as the development of higher-order macrostructure (Coombs et al, 1981). It has been suggested and a fair body of research supports the claim that the development of higher-order macrostructure or Hoc's "SRT's" can be facilitated through the provision of appropriate concrete and analogical conceptual models (Du Boulay and O'Shea, 1978; Du Boulay et al, 1981; Mayer, 1975, 1976, 1981; Miller, 1974; Sime et al, 1977). The next section expands upon the theme of language instruction by introducing research that investigates computer programming as problem solving.

Computer Programming as Problem Solving

Card, Moran, and Newell (1983) present an interesting developmental theory of cognitive behaviour that raises several pertinent questions about the relationship between programming and problem solving in an educational environment. Situating their theory within that of contempory information-processing psychology and especially in terms of coginitive skill acquisition (Anderson, 1980, 1981), they contend that all cognitive behaviour is located and progresses along a continuum from problem solving to cognitive skill. Problem solving behaviour will with practice become cognitive skill, and cognitive skills in turn start out in problem solving behaviour.

They state that all cognitive behaviour can be located within a three-dimensional matrix which comprises a task dimension representing specific task domains, a human processing dimension reflecting the perceptual, cognitive, and motor structures of the human processor, and a skill dimension. The skill dimension indicates the degree of skill with which the behaviour is performed. It defines a continuum ranging from problem solving activity towards cognitve skill. Card, Moran, and Newell (1983) emphasize that problem solving behavior is simply the less-skilled end of this continuum. It is not a distinct category of behaviour. They argue that problem solving behaviour will with practice become cognitive skill, the determining variable being the amount of search control knowledge through a problem space available to the learner during the performance of the activity.

Thus a novice programmer asked to solve even an elementary looping problem will exhibit behaviour which is characteristically of a problem solving nature. He/she will search through the problem space state by state determining whether each successive state is approaching the goal state, perhaps backtracking , pausing, or searching via trial and error along the way. Solving the problem becomes the process of finding a sequence of operators (or a path) that will transform the initial state into the goal state.

An experienced programmer on the other hand would be more likely to exhibit skilled behaviour in deciding upon an appropriate sequence of actions because he/she has available what Card, Moran, and Newell (1983) term "methods" or "packaged operator sequences". The decisions become non-problematic because the experienced programmer does not have to search through the problem space evaluating the result of the application of each operator. Such packaged operator sequences could perhaps be compared to Hoc's SRT's (1977), or to Shneiderman and Mayer's (1979) semantic knowledge constructs to be discussed shortly which consist of higher-level programming constructions independent of specific language. All of these models share an implicit developmental view of learning which in terms of programming views the knowledge state of the learner progressing from lower-level detailed syntax towards increasingly higher-level semantic knowledge.

The developmental view of Card, Moran, and Newell (1983) sees most cognitive activity developing from problem solving behaviour. If this view is considered in terms of applied educational strategy, it suggests that the learning of a new cognitive skill is initially at any rate characterized by problem solving behaviour. The learning of programming could be viewed as a form of probelm solving characterized by the specific task domain involved. This is in accord with recent strategies towards the teaching of programming as problem solving that aim to develop not only programming skills but general problem solving abilities as well (Kelman et al, 1983; Shneiderman, 1980; Papert, 1980).

Traditional views of programming and the teaching of programming have concentrated primarily on the acquisition of a specific language and its subsequent use in performing routine tasks such as the generation of consecutive integers, the listing of multiples, and the generation of fibonacci numbers (Kelman et al, 1983; Weinberg, 1971). A programming language was taught much like a foreign natural language in that students first learned the syntax of the language and then practiced on simple exercises. Such an approach was found monotonous by many students and teachers because of its emphasis on low-level cognitive skills. As has been suggested, many educational researchers and curriculum designers now view programming as a problem solving activity (Kelman et al, 1983) in which the process not the end-product is important. Students work within problem solving environments such as the Logo "microworlds"

(Papert, 1980), or use simulation programs in which they can explore and solve historic and real world problems. They may also design and validate computer models thereby developing group cooperation and data organization skills. The ability to model problem situations allows the student to explore problems and not simply to obtain answers. Such problem explorations might reveal a range of possible solutions and in so doing de-emphasize the import of the single correct answer.

Kelman et al (1983) support their view that the teaching of programming is the teaching of problem solving by stating that to write effective programs the student must first explore the problem situation, define the problem, and work on various solutions until finding the optimal one. The joy of programming the authors conclude as a problem solving activity is that when the student has solved the problem he knows it; the program runs.

Brooks (1983) defines computer programming as "a set of problem solving tasks in a semantically rich domain..." (Brooks, 1983, p 543). He hypothesizes that programming is a process of constructing mappings from a problem domain through intermediary domains and into the progamming domain. Comprehending a program involves the construction of part or all of these mappings.

Shneiderman and Mayer (1979) present a cognitive model of programmer behaviour and problem solving that separates syntactic knowledge from semantic knowledge and emphasizes the internal semantic representation created by the programmer in such activities as program composition, comprehension, debugging, modification and learning. Their model descibes these activities in terms of:

1) The cognitive structures developed by programmers;
2) The cognitive processes involved in accessing or adding to these structures.

Using as their basis an information processing model of cognition in a fashion similar to Card, Moran, and Newell (1983), they describe a multi-leveled cognitve structure stored in long-term memory comprising programming concepts and techniques. They divide this structure into two components: semantic knowledge and syntactic knowledge.

Semantic knowledge consists of general programming concepts independent of specific language. At its lowest level, it conceptualizes simple statements and structures (eg. assigment statements, subscripted arrays, data types). At its higher levels, it assists in problem solving by making available to the programmer meaningful sets of information.

Syntactic knowledge concerns the details of specific languages (eg. form of iteration, names of library functions). It is the specific instance of the generalized semantic representation.

Semantic knowledge is acquired through meaningful learning including problem solving and instruction which encourages the learner to assimilate new concepts within existing semantic or "ideational structure" (Shneiderman and

Mayer, 1983, p 223). Syntactic structure is stored during rote learning which does not integrate it into existing cognitive structure. Shneiderman and Mayer draw parallels between their semantic/syntactic continuum and those of Polya (1957): know how/know what, Greeno (1973): algorithmic/propositional knowledge, and Ausubel (1968): rote/meaningful learning. It is apparent that they have much in common with Ausubel, especially Mayer whose work closely parallels and extends that of Ausubel into the realm of the application of concrete models as advance organizers.

In examining the processes involved in problem solving tasks, Shneiderman and Mayer use Polya's four-stage paradigm of the problem solving process:

1) Understanding the problem in defining the initial and goal states;
2) Devising a plan or a general strategy of solution;
3) Operationalizing the plan;
4) Verifying the solution to determine if it in fact works.

In the composing of a program, a problem is analyzed in working memory in terms of the definition of initial and goal states with input from information (both semantic and syntactic) from long-term memory. Next a general plan is formulated in terms of broad programming strategies termed "internal semantics" which involves a funneling down of the internal representation to more specific details and finally to the actual code. The development of this internal semantics can proceed in several manners:

1) A top-down approach whereby the problem solving activity progresses from the general goal to the specifics;
2) A bottom-up approach in which low-level code is generated first in an attempt to build up to the goal;
3 A structured-programming approach involving modularization and the use of sub-goals.

Each of these techniques leads to a funneling of the internal semantics from a general to a specific plan. Shneiderman and Mayer conclude that once this internal semantics has been developed, the writing of the program itself is straightforward.

Sheiderman and Mayer distinguish between two teaching/learning strategies:

1) The classical syntactic approach focusing on detail and statement validity;
2) The problem solving approach concentrating on high-level language-independent semantic knowledge.

The second approach parallels that of Kelman et al (1983) who as, previously mentioned, view the teaching of programming as the teaching of problem solving itself. However, Shneiderman and Mayer conclude that each approach has some validity, and that a combined approach is probably desirable. Semantic knowledge is essential for problem analysis while syntactic knowledge is applied during the implementation phase.

They present the conclusions of several experiments which formed the basis of their model. One such study (Shneiderman, 1976) tested statement order as a determinant of memorability. Two short FORTRAN programs, one of which was printed in logical order and the other of which was shuffled and listed, were presented to novices and experienced programmers. The novices did poorly in recalling the program in both versions whereas the experienced programmers faired much better with the proper executable program. Shneiderman concludes that the experts converted the code into more general internal semantics and were thus able to recall the executable program from long-term memory far more effectively than the novices who did not possess the same degree of prior meaningful knowledge.

Other experiments tested for the effects of commentary and mnemonic variable names in addition to the use of modular program design and flowcharting. It was found that programs using comments and those using mnemonics were easier to comprehend, that modular design produced more comprehensible programs (though group differences are reported significant at the .08 level), and that flowcharts aided students in problem solving tasks in some situations and hindered them in others (Shneiderman et al, 1977). Students who had been previously exposed to flowcharts were aided by the device and the authors conclude that in these cases the flowcharts helped in the translation process from syntax to semantics. In other cases the flowchart possibly acted as an alternative syntactic representation and actually hindered the development of internal semantic structure.

Green (1980) argues that choosing a good mental representation for a problem definitely improves the probability of finding a solution. He cites the work of Mayer (1975, 1976, 1978, 1981) as evidence of this in the field of computer programming. Contrary to Shneiderman and Mayer (1979) however whose theory this proposal has just discussed and to Mills (1975), Green contends that programming is not so much a top down or bottom up process, as it is a leap of faith towards a final goal followed by a verification process to construct and solidify a proper path. He quotes Polya (1971) who claims that formal exposition is fine for checking the argument but it is often directly opposed to the manner in which mathematicians arrive at a proof. Intuition in other words will sometimes rush ahead of formal reasoning. Green adds that programs are not necessarily written in a top-down manner; they merely look as though they were.

Hartley (1978) presents two approaches to the teaching of problem solving skills with the computer. The first involves the inclusion of simulation facilities within author language tutorial programs. He provides an example of the CALCHEM Project at Leeds University in which students interact with a program that analyses nuclear magnetic resonance spectra. The other approach involves having the student put his solution in the form of computer programs. The advantage of this approach, Hartley contends, is that solving problems algorithmically requires a clear understanding of what is required because the student is forced into giving a step by step solution. Hartley endorses the use of Logo because it is able to provide a convenient grammer for representing mathematical processes. As will be apparent shortly, Logo has been and is currently being implemented in just such a capacity.

Soloway et al (1982) agree with Hartley that viewing a problem in an algorithmic manner will facilitate its solution. They conducted several experiments with college freshman in order to determine the validity of their hypothesis that programming does enhance problem solving ability because it encourages the required procedural view. Groups of students were asked to solve algebra word problems either directly through written equations or algorithmically by writing short programs in BASIC. Their results indicate that groups asked to solve the problems algorithmically did do significantly better. Soloway et al (1982) conclude that programming can enhance problem solving ability because of its emphasis on debugging strategies and the practice of decomposing a problem into explicit sub-steps.

Kelman et al (1983) argue that programming is a creative activity in which students need not master any one computer language but need to understand the underlying structure of all programming in order to be able to utilize new programming tools when required. This view is similar to that of Weinberg (1971) who states that it should be the goal of formal education to train programmers to the extent that they can use programming tools as tools to further their learning. Weinberg criticizes educational institutions for concentrating on the teaching of specific languages and not on generalizable principles of programming design that students can use over a broad range of differing languages.

Problem Solving with Logo

Logo was developed to provide a framework for the teaching of mathematical concepts. Papert (1971a, 1972b) has expanded the language in creating a learning environment within which students of all ages can take part in an active learning experience within what Papert terms "microworlds" (Papert, 1978). A microworld is a "subset of reality or a constructed reality whose structure matches that of a given cognitive mechanism so as to provide an environment where the latter can operate effectively" (1978, p 1). Papert admits that such a concept is not new. What is novel is the availability of a microcomputer technology that can make of microworlds a refined, systematic and theory-based branch of education.

Papert views technology as offering great potential benefit to education in terms of allowing students to manipulate, extend, and apply it to personal projects. He places himself within the tradition of Dewey, Montessori and Piaget (Papert, 1971a, 1971b, 1973) in that he believes that children learn by doing and by thinking about what they do. The use of microworlds provides a model of a theory of learning in which the student is actively engaged in exploration within an environment sufficiently bordered and transparent for constructive exploration and yet sufficiently rich for significant discovery. Papert describes his turtlegraphic microworld as providing "hooks" on one side to preformal intuitive body knowledge and on the other side to formal rigorous mathematical knowledge. The child learns the formal concept of circle for example by playing turtle and walking about in a circle, before engaging the Logo turtle to do the same. This type of learning Papert refers to as body or ego syntonic, as opposed to the dissocciated learning found in the traditional classroom. Papert states that a child's "intellectual growth must be rooted in his experience" (Papert, 1971b, p 4).

He has operationalized his theories in association with colleagues at the MIT AI laboratory by creating Logo environments in subject areas as diverse as geometry, music, physics and biology. In this sense, Logo environments have been designed to allow children to write musical compositions (Bamberger, 1972), to think about relations in temporal and tonal space (Papert, 1973), to simulate a body in Newtonian mechanics (Papert, 1978b), and to simulate a system in stress mechanics (Abelson and DiSessa ,1976). Evidence gathered at the MIT laboratory (Papert, 1971a, 1971b, 1976; Solomon, 1976; Papert and Solomon, 1972) has indicated that children do acquire programming and problem solving skills within Logo environments, although much of the evidence has been anecdotal. Researchers are now beginning to criticize much of the extant Logo data because of the lack of an objective methodology (Krasnor and Mitterer, undated).

Howe and Ross (1981) have experimented with Logo in the secondary school mathematics program, using the language to simulate a mathematical system. Students work towards understanding a mathematical problem by building and experimenting with a model of the process for solving that problem expressed as a Logo program. The authors compare the Logo tool kit of mathematical parts to the Mecanno construction kits with which children used to construct physical models. Rather than building and running physical models made from strips of metal, the student builds and runs symbolic models of mathematical mechanisms. They have been using Logo in this work because of its library of powerful primitive instructions with which the student can construct personalized procedures to help solve specific problems.

Howe and Ross (1981) continue the Mecanno/Logo parallel by enumerating what a student must learn in order to manipulate Logo as a modeling tool:

1) He must become familiar with the syntax;
2) He must learn how to assemble components into procedures;
3) He has to learn about the structures and mechanisms being modeled;

4) He has to understand basic mathematical concepts;

5) He has to learn how to break complex constructions into sub-constructions;

6) He has to learn to cope with mismatches between planned and actual behaviour in terms of program debugging.

Though they admit that the above is a formidable teaching task, the authors endorse a structured teaching approach that introduces a mathematics or programming topic, provides specific procedures, and gives exercises which can be solved using these procedures. Logo allows, they insist, for the tailoring of instruction to the ability of the individual student. In this sense, a less able student can be given a sample procedure as a "concept demonstration" of a particular mathematical problem with which he will be able to acquire some insight into the underlying processes (eg. relationship between shapes of different polygons).

Evaluation studies conducted on 11-13 year old boys concluded that the teaching and use of Logo did improve marginally performance on algebra compared to students who had not received this instruction. Consultation with teachers yielded the information that the experimental group pupils could argue sensibly about mathematical concepts and explain mathematical difficulties clearly. They rated the control group poorly on these abilities (Howe and Ross, 1981).

Du Boulay and Howe (1981) have experimented with providing mathematically-weak student teachers with exposure to the writing and debugging of Logo programs. They conducted two evaluaton studies. In the first, 15 volunteers were recruited and in the second, 21 conscripts were used. The first study emphasized the learning of Logo as a programming language, and the subsequent solving of specific mathematical problems. Those students who persevered and learned the language using Logo to explore mathematical areas in which they had had difficulty were very pleased with their personal progress in overcoming their difficulties.

In the second study, students spent far less energy in the writing of Logo programs. They were given pre-defined procedures and discussed the mathematical relationships exhibited by them.

Results of pre and posttests showed slightly better performance for those who had worked with Logo as opposed to a control group. Du Boulay ahd Howe (1981) do state that the small cell numbers and high within group variance make comparisons rather difficult. Because students in the second study did not have the opportunity to write their own programs, they showed less involvement with their work and made fewer mathematical discoveries. The authors conclude, however, that there is a trade-off between the benefits of problem solving with the computer and the extra time necessitated in learning a programming language.

Chait (1978) studied the problem solving activity of five seventh-grade students in a graphics-oriented Logo environment. Chait distinguishes her work in terms of its methodology which contrary to most Logo research using either a case study or experimental approach, relied on a form of protocol analysis derived partly from Newell and Simon (1972). Recording interactive computer sessions over a ten-week period in "dribble files", the author developed a classification system with which she was able to identify various stages students passed through in the learning of Logo. In addition, she used the dribble files to classify program bugs and the debugging action taken as a result.

Utilizing Polya's classification scheme, the author draws parallels along with other researchers (Statz et al, 1973) to problem solving in a Logo environment. The child decides on the nature of the problem, devises a plan in terms of sub-goals and sub-procedures, carries out the plan by writing the actual code, and debugs the program if it does not run as expected. Debugging is assumed to take place in Polya's fourth or verification stage (Statz et al, 1973). The Chait study, however, contradicts this assumption in that debugging activity was found to occur primarily in Polya's second or planning stage before the program itself had been written. Brown and Rubinstein (1974) refer to this as "bottom-up" debugging.

Chait's classification scheme, which the author admits is merely a first step in the analysis of children's experiences in the Logo environment, allows for the representation of program structure in procedural nets along the lines developed by Feurzeig and Lukas (1975) Such nets revealed both horizontal programming styles, in which a program was revised many times, and vertical programming styles with few revisions and a deep hierarchical structure. The author states that style tended to develop from horizontal to more vertical programming with an accompanying increase in the use of sub-procedures.

The classification system also revealed relationships between the type of bug encountered by the student and the subsequent debugging action taken. Two conclusions were reached:

1) Several debugging actions (eg. reversing a command; executing a procedure line by line) were specific to certain bugs;

2) Only one debugging action (clearing the screen and starting over) was used with a wide range of bugs.

Pre and posttests including basic arithmetic skills and a set of geoboard problems (duplication, rotation, mirror-imaging) were administered to the five students. No significant differences in scores were revealed.

Chait concludes by admitting that the design and implementation of her classification scheme suffered from a number of shortcomings not the least of which was the lack of reliability in both the collection and interpretation of session protocols. Because there was no external observer, the objectivity of the dribble file annotations is brought into question. Further research, the author states, should include reliability tests on collection and annotation of the protocols.

Krasnor and Mitterer (undated) criticize Chait in addition to other researchers (Statz et al, 1973) for a lack of objective measurement. In reference to the Chait (1978) study, they argue that the small sample size and the lack of a control group make the results difficult to interpret. In addition they claim that the test battery used was not tied to any theory of problem solving. They critisize Statz et al (1973) who investigated problem solving with grade 4 students in a Logo environment for altering their training procedures partway through the study, and for also not tying their measures of problem solving ability to any extant approach (Polya, 1965; Newell and Simon, 1972). Krasnor and Mitterer claim that the entire Logo literature merits criticism from an experimental point of view because of the general lack of objective methodology (p 9).

They question the contention that the Logo environment facilitates the learning of skills and heuristics which transcend the immediate task environment. Such skills as the breaking down of problems into sub-problems, the systematic planning of actions to achieve goals, and the use of debugging in the successive refinement of problem solutions have not been, the authors claim, experimentally tested in the literature in terms of general transfer to other problem solving domains. The general transfer of these "powerful ideas" (Papert, 1980) requires some sort of congruity in terms of processes and knowledge between learned and new tasks, in addition to an awareness by the learner that the new problem situation is similar to one previously encountered. As well, the degree of transfer may depend on the completeness of the original learning experience and an exposure to a variety of situations in which the skill is usefull.

Logo experiences, they conclude, have typically not encouraged the skills needed in order to recognize correspondences between problem situations, nor have they exposed students to a wide enough variety of situations to promote transfer (p 5). They continue that because the Logo literature consists primarily of testimonials (Goldenberg, 1980; Watt, 1982), curriculum guides (Birch, 1980), manuals (Abelson, 1980), and studies presenting anecdotal evidence, there is no solid evidence that any of the powerful ideas generalize to other domains.

However, they do enumerate several of the language-dependent and environment-dependent components that could potentially lead to such transfer of problem solving skills. The language dependent components include the turtlegraphics aspect of Logo which allows the child to "concretize the abstract", as well as the ease of writing structured programs which facilitates sub-goal and means-end analysis.

Environment-dependent components include the stress on play and exploration and the small group nature of the Logo environment which encourages social interaction. Such interaction may foster learning as children observe and tutor each other.

Leron (undated) disagrees with the connection between spontaneous non-directed "piagetian" interaction with the computer and the acquisition of Papert"s "powerful ideas" (Papert, 1980). He contends that under such non-directed conditions even bright children fall into a "hacking" kind of programming characterized by trial-and-error activity which is accompanied by little planning or reflection. The development of powerful ideas and new intellectual structures requires, Leron states, " a bigger push" in terms of more planning and directing of the student's activity.

In addition, Leron suggests that while most children acquire quite easily the first steps of Logo programming (eg. navigating the turtle, writing simple procedures), they encounter a great deal of difficulty with the more complex activities related to the use of subprocedures and structured programming in general. He distinguishes between two levels of learning structured programming:

1) Learning the syntactical rules whereby procedures call each other;

2) Creating a conceptual framework in terms of conceiving a complex procedure as a hierarchy of subprocedures with interfaces between them.

The first level poses no difficulties for most children, Leron observes, but the children who he has encountered have resisted the suggestion of using subprocedures and continue with a linear style of programming. One source of difficulty, he suggests, is the lack of a clear concept of the interface between two subprocedures, and the importance of the turtle state before and after each subprocedure.

Leron concludes that the brief duration of most Logo courses does not encourage the development of a deeper conceptual understanding of such concepts as subprocedures, variables, and recursion. He suggests a well-planned spiral course extending over several years based on a better understanding of children's learning in the Logo environment.

The following section will discuss research conducted into providing this deeper conceptual understanding through the provision of conceptual models of computer systems and languages to novice programmers and students. If computer programming is viewed as a problem solving activity, then the development of programming skills should transfer to the development of more general problem solving abilities. This is, however, as has been observed, a point of contention. Kelman et al (1983), Papert (1980), Green (1980), and Howe and Ross (1981), have argued for some degree of transfer, but as Howe (1979) cautions there is not much hard evidence supportative of this contention. Presumably this remains an area in which further research is required.

Conceptual Models

Systems designers and applied psychologists are becoming more convinced that people deal with complex interactive systems by utilizing a conceptual model of the device (Young, 1981). This notion of "conceptual model" is rather poorly defined but one of its critical assumptions is that the user will adopt some representation or analogy to assist him in understanding and interpreting the system's behaviour.

The purposes of a conceptual model, according to Myers (1980), are several:

1) It allows the user to predict the effects of commands;
2) It ties together otherwise seemingly independent procedures;
3) It enables the user to determine the reason for error conditions;
4) It serves as the basis for new approaches to working with the system.

Myers suggests that effective conceptual models should be constructed from concepts familiar to computer scientists but cast in non-technical language. Just what the model should contain, he admits, remains to be investigated, but concepts such as context (executing one command set at a time), files, agency (agents within systems performing different functions), and operations (copying and storing) are factors that he argues should be included.

Craik (1943) suggested that people construct mental models in order to represent objects in the environment. This review has briefly touched upon the question of user models in the discussion of the notional machine (Du Boulay et al, 1981) which will be elaborated upon shortly.

Research into the application of conceptual models has been directed towards educational as well as other, more commercial interests. Young (1981) describes designs for conceptual models of hypothetical pocket calculators. He distinguishes between models which merely offer "cover stories" or simple descriptions of the calculator's behaviour, and those which illuminate the relationship between actions taken by the user and the tasks the calculator carries out. This latter class of models is divided into three levels or "arenas": an action, a task and an abstract machine arena, each of which is a distinct psychological representation of the calculator's behaviour. Young concludes that an effective interface must include such a conceptual model and that future research is required in order to better establish what is meant by the term "user conceptual model".

Moran (1981a) has designed what he labels a "Command Language Grammer" (CLG) which is a representation for the user interface of interactive computer systems. He defines the user interface as consisting of aspects that the user comes into contact with, physically, perceptually, or conceptually. The major thrust of his work involves the determination of heuristics to guide effective system design. Moran's grammar is divided into three components: a conceptual, a communication, and a physical component. For the purposes of the present discussion, only the conceptual component is of interest. This component contains the abstract principles around which the system is organized. It is divided into a semantic level and a task level. The task level analyzes the user's needs and structures his domain in a way compatible with the system. The semantic level provides the linkage between the data structure and procedures of the system, and the user's conceptual model of these structures and operations. It lays out the conceptual model of the system appropriate for accomplishing specific tasks.

Moran views his CLG as a psychological model of the user's knowledge of a system. In this sense, the three components form a hierarchy which can be used to determine instructional precedence leading to the most effective learning. Moran states that the user should learn the system in the order determined by this hierarchy. Thus, the physical commands within the physical component cannot be mastered before the acquisition of the more general concepts within the conceptual component.

Of specific interest to the present discussion, is the question of learning by analogy to appropriate concrete models. Such analogs must describe both the analog and abstract concepts and the mapping between them. Moran remarks that, " It is often stated that systems are easiest to learn by analogy with some appropriate physical system " (Moran, 1981a, p 42).

Moran states unequivocably that the purpose of his model is to force the designer to create a conceptual model of the system during its design for the user to subsequently assimilate. He defines such a model as, " The whole conceptual organization of the computer system from the user's point of view" (Moran, 1981b, p 5). He emphasizes that the model is an integral part of the user interface, and declares that it can be assimilated by the user and that such assimilation in most cases requires explicit training and/or documentation. He sees two problems associated with the assimilability of conceptual models:

1) The model might be so abstract that it proves difficult for people to grasp;
2) The model, because of its possible simularities to the user's existing models, might create interference and confusion between itself and prior knowledge.

(Moran, 1981a)

Moran states that, "It is a task for psychology to discover such problems and to develop techniques for evaluating conceptual models, so that assimilable models can be reliably designed" (Moran, 1981a, p 42). He concludes that computer systems with display devices have the potential for visually representing the conceptual model, thus making it easier to learn. The area of the graphic representation of computer systems and languages remains one in which further research is definitely called for. The section on graphics and text in CAL reviews briefly the small body of research addressing the application of graphics within a CAL environment.

Jagodzinski (1983) concurs with Moran, and states that a conceptual model of the system pitched at an appropriate level must be provided to the novice user. He adds that this model should represent the structures and operations of the system to the level of detail as it is within the novice's power to affect. This view parallels that of Du Boulay et al (1981) whose notional machine is, similarly, a simplified representation of the system. Jagodzinski endorses the use of system overview screens to assist the user in orienting and navigating himself within the particular system.

Howe and Ross (1981) believe that a novice's ability to learn programming is a function of how developed is his mental representation of the machine he is trying to use. If he lacks such a mental representation, his progress will be impeded. They define such a representation as a description of the machine pitched at a level of detail appropriate to the needs of the student. In this respect, the detailed division of the CPU into an arithmetic and a control unit is considered unnecessary for the novice Logo programmer. This simplified description of the language's operation within the machine is called a "virtual machine", the purpose of which is, "to provide a model as a context for introducing programming concepts, for interpreting the machine's responses, and for establishing a small volcabulary for talking about programs and the activity of programming" (Howe and Ross, 1981, p 96-97). Such a virtual machine could perhaps be compared to the concept of the "notional machine" (Du Boulay et al, 1981) previously described, or the concept of "conceptual window" (Sime and Fitter, 1978) into the system's underlying processes. Such conceptual or concrete models of the underlying or essential processes of a system have resulted in improved understanding and performance both with children in a Logo environment (Du Boulay and O'Shea, 1978; Du Boulay et al, 1981) and with older students at the secondary and university levels (Mayer, 1975, 1976, 1979, 1981; Miller, 1974; Sime et al, 1977). The work of Mayer, in particular, will be discussed shortly.

Du Boulay et al (1981) expand on their concept of the "notional machine" by defining it as an "idealized conceptual computer whose properties are implied by the constructs in the programming language employed" (p 237). The properties of this machine are not hardware specific but language specific. One can therefore distinguish between a BASIC machine and a Logo machine. The authors enumerate two principles inherent in an effective instructional strategy based on a notional machine for making the hidden operations of a language more transparent to a novice:

1) The notional machine should be conceptually simple;
2) The novice should have at his disposal methods for

observing the machine in action.

(Du Boulay et al, 1981, p 237)

The authors refer to these principles as simplicity and visibility, respectively.

Simplicity includes functional, syntactic, and logical factors. Functional simplicity is dependent on the limitation of the set of allowable "transactions" (Mayer, 1979) which consist of operations that characterize what a language does in terms of a simplified functional model. They are neither a description of the hardware nor a formal semantic definition of the particular language. Functional simplicity is distinguished from both syntactic and logical simplicity. Logical simplicity intends that problems of interest to the novice can be explored by simple programs. Syntactic simplicity is ensured via uniform rules for writing code and a minimal number of special cases to remember. It is functional simplicity, however, that has the greatest influence on the learnability of the notional machine.

Simplicity may be optimized by restricting both the language to a few constructs and the number of possible actions taken by the machine. Such a strategy has been adopted in the Logo turtlegraphic environment Du Boulay et al (1981) endorse the use of Logo because its procedural nature and unrestricted naming permit novices to produce interesting results with simple programs.

Attempts toward visibility, toward making aspects of the notional machine more accessible to the novice, include pictorial or written traces that can comment on the actions taken by the notional machine during execution of a program. In addition, slot boxes (Hillis, 1975) have been designed into which tokens are placed having printed and machine readable instructions. The tokens may be arranged in any sequence to alter the order of commands, and the flow of control is illustrated by a succession of bulbs beside each slot.

Du Boulay et al (1981) describe an implementation of Logo called ELOGO with which they have operationalized their concept of notional machine. They use a simple button box containing turtlegraphic commands and a facility for defining procedures. The button box concretizes the notional machine and is used as a foundation to build the novice's understanding of the more complete ELOGO system. It introduces in a manner comprehensible to the novice such concepts as command, argument, procedure, sub-procedure call, and recursion. An effort was made in the design of the model toward both functional simplicity and visibility. The basic program unit is the procedure and a simple filing system was designed allowing for easy storage and retrieval. Visibility was promoted by externalizing most of the hidden actions (eg. storing a procedure) via written comments from the system. Documentation and teaching materials were designed in conjunction with the implementation of the language itself to ensure that comments from the system such as error messages could be worded using the same analogies as those used in the teaching materials.

Mayer (1981) agrees that the ELOGO model meets the specifications of simplicity and visibility in that it is a simple, familiar model of the operations involved in Logo. He states that, " It allows the user to develop intuitions about what goes on inside the computer for each line of code" (p 126). However, he argues that the researchers have not provided empirical evidence to substantiate the claim that the Logo machine model actually influences the problem solving abilities of novices as compared to more traditional instructioal methods emphasizing only hands-on experience.

In teaching children to program in Logo, Du Boulay and O'Shea (1978) have proposed various ways of revealing to them the underlying processes of the system, using plotting devices, turtles, and tune boxes. They state that when the child is simply using Logo at top level to draw various shapes, it is sufficient to present the computer as a drawing machine. However, once the child begins to use subroutines and variables, once he is working with Logo as a programming language, this analogy can actually hinder his performance because it cannot by its very nature provide a conceptual window into what the program is doing.

Du Boulay and O'Shea (1978) discovered that in order for the children to learn and apply Logo effectively, they had to be given a model of how the computer operated in terms of breaking down the system into its functional components (eg. workplace, permanent store). The different activities performed in Logo including issuing commands and running procedures were described in terms of information transfer between the various functional components. It was found that effective learning occured only when the Logo system presented during instruction was consistent with that presented by the model.

The design and provision of conceptual models is considered an effective instructional strategy by many researchers (Myers, 1980; Moran, 1981; Howe and Ross, 1981; and Du Boulay et al, 1981) who view the model as a means of concretizing abstract, conceptual information. Such models can simplify the sometimes confusing and unconnected detail of syntax, and make more comprehensible the higher-order macrostructures of a programming language. The following section reviews the evidence for and against advance organizers in preparation for the discussion on the application of conceptual models as advance organizers.

Advance Organizers

The fortunes of advance organizers have proved less than spectacular in the twenty years since they were introduced by David Ausubel and his colleagues (Ausubel, 1960). The educational benefits first touted by their developers have been questioned or overtly disputed by some (Barnes and Clawson, 1975), extended and refined by others in new applications (Mayer, 1975, 1976), and religeously defended both by the originator (Ausubel, 1978), and newer disciples (Lawton and Wanska, 1977). Whatever one wishes to conclude personally about these contentious viewpoints, the positive results obtained from dozens of studies supporting the use of advance organizers do argue for the consideration of Ausubel's early theory in current research and development.

Ausubel has defined the advance organizer as a deliberately prepared set of ideas presented at a higher level of abstraction, generality, and inclusiveness to the learner in advance of meaningful learning. He emphasizes the salient differences between organizers and overviews, arguing that the latter present material at the same level as the instruction itself (Ausubel and Robinson, 1969), whereas the organizer, presented in advance of instruction, provides, "relevant anchoring concepts" to which new less inclusive material may be subsumed. Ausubel states that the advance organizer allows the learner to, "exploit his existing knowledge as an ideational and organizational matrix for the understanding and fixation of new material" (Ausubel and Robinson 1969, p 57).

In this sense, advance organizers differ from overviews in that they are relatable to ideational content in the learner's current cognitive structure (Ausubel, 1963, 1968). One might conceptualize the advance organizer as an operational bridge between a learner's existing knowledge and the new material he is to assimilate.

In Ausubel's view, meaningful learning occurs when an idea is related in some sensible fashion to ideas already possessed by the learner Meaningful learning must exhibit the properties of substantiveness, that is, a robust relationship to already existing cognitive structure, and non-arbitrariness, which implies that the relationship between new knowledge and existing cognitive structure (eg. between "equilateral triangle" and "triangle" in general) is a relationship of specific instance to general case, and not an arbitrary mapping of the learning task to existing cognitive structure. For effective learning to occur, the learner must also possess psychological meaningfulness, that is, he must have both the relevant ideas to which to relate new material and the intent or motivation to do so.

Rote learning, in contrast to meaningful learning, is the process in which new material is assimilated in an arbitrary unconnected manner. Ausubel contends that learning will be increasingly rote to the extent that material lacks logical meaningfulness, and the learner lacks both the relevant ideas in his cognitive structure and a meaningful learning set.

Ausubel's subsumption theory makes two assumptions about human cognitive processing:

1) It is less difficult for human beings to grasp differentiated aspects of a previously-learned, more inclusive whole, than to formulate the inclusive whole from its previously-learned parts;
2) An individual's organization of the content within a subject domain consists of a hierarchical structure in which the most inclusive ideas occupy a position at the apex of the structure and subsume progressively less inclusive and more differentiated propositions, concepts, and facts.

(Ausubel and Robinson, 1969, p 168).

From these assumptions, which have been seriously questioned by other researchers (MacDonald-Ross, 1978), Ausubel concludes that information will be more effectivley learned and recalled when more inclusive and relevant ideas are already available in cognitive structure to serve a subsuming role.

The critics of Ausubel's theories have been numerous. Some have directed their attacks against his theoretical base directly (MacDonald-Ross, 1978), while others have criticized him for a lack of an operational definition of the term "advance organizer" (Hartley and Davies, 1976; Barnes and Clawson, 1975).

MacDonald-Ross (1978) argues that for a concept to be made operational, deep conceptual analysis must be done. He attacks Ausubel's theory in terms of its assumptions about the hierarchical nature of cognitive structure, stating that neither the structure of material nor the process of cognition can be adequately modeled as a hierarchy. He champions modern theories of cognition in which the "heterarchy" is the logical structure of cognition (eg Pask, 1975, 1976).

Barnes and Clawson (1975) reviewed 32 studies in which advance organizers were evaluated They found that out of the total, 20 studies did not support the use of organizers. Among these were studies by Schulz (1966), Woodward (1966), Barron (1971), Barnes (1972) and Clawson (1972). Among those which did support the use of organizers are several studies by Ausubel and his colleagues (Ausubel, 1960; Ausubel and Fitzgerald, 1961; Ausubel and Youssef, 1963; Weisberg, 1970)

The Weisberg study is particularly interesting in that it used three types of organizers, two of which were visual; the first, a graphic of the North Atlantic ocean floor, and the second, a map of the ocean floor. As Barnes and Clawson (1975) note, studies of operationally defined non-written organizers are rare, and more research, they suggest, should be directed toward this area. The Weisberg study found a significant improvement in learning for the two visual organizers against a control group that recieved no organizer

Hartley and Davies (1976) do not disagree theoretically with Ausubel. They state that organization is the hallmark of good teaching, and that the sequence and arrangement of material influence not only what is learned but attitudes toward the usefulness and importance of what has to be achieved. Any procedure, they argue, that makes this organization more obvious is likely to facilitate the learning of meaningful material.

The major reservation expressed by Hartley and Davies (1976) is the lack of an operational definition in terms of recognized procedures for designing advance organizers. They suggest a functional rather than an operational definition, but this is not a practical solution, they emphasize. This lack of operational definitions is expressed by others in the literature who do not necessarily disagree with Ausubel on theoretical grounds.

Ausubel has responded to criticism, in a defence of advance organizers (Ausubel, 1978), in which he directs detractors to read his books and articles in order to obtain precise operational criteria for an adavance organizer, and a discussion of how to construct one. However, he contradicts himself by cautioning that apart from describing organizers in general terms and providing an appropriate example, one cannot be more specific because such detail is subject and learner specific.

Mayer and Bromage (1980) expand upon Ausubel's original definition of advance organizer by including concrete and physical analogies in addition to abstract stimulii. They define an advance organizer as a stimulus:

1) Presented prior to learning;

2) Containing a system for logically organizing the incoming information into a unified structure.

While the organizer may, in fact, be abstract or concrete, it must be able to logically integrate new information. Mayer and Bromage justify this expanded definition by citing research which indicates that concrete analogies may serve as advance organizers under certain conditions (Royer and Cable, 1975, 1976; Mayer, 1975, 1976). A discussion of this research follows in the section investigating the use of conceptual models as advance organizers.

Mayer (1979a) states that organizers do not necessarily always lead to improved learning. Both Ausubel's subsumption theory and Mayer's assimilation theory propose a significant effect if and only if:

1) Learners would not have had prior knowledge subsumers available during learning;

2) Learners did have these but would not have ordinarily used them.

Mayer concludes, therefore, that the best test of the effect of advance organizers is when material is unfamiliar, technical, or otherwise difficult for the learner to relate to his or her existing knowledge. He adds that organizers are always relative to a particular learner and subject matter, and that further research is required to determine the best analogies, images, and examples to serve as effective organizers for specific learners and subject areas.

Mayer (1979b) further enumerates six conditions necessary for advance organizers to have any significant effect:

1) Material must be unfamiliar; it should not ellicit or contain a general subsuming context;
2) Material must be potentially meaningful or conceptual;
3) The advance organizer must provide or locate the meaningful context;
4) The advance organizer must encurage the learner to use that context;
5) Learners do not possess relevant conceptual contexts;
6) Evaluation must measure the breadth of learning in terms of transfer learning and long-term retention.

Ausubel (1978), similarly, proposes the necessity for appropriate evaluation in terms of measuring the breadth of learning. He argues that negligable results have been found in advance organizer studies which evaluate simple recall and recognition without measuring performance in terms of transfer to novel situations and long-term retention. Mayer has conducted numerous studies addressing these aspects of evaluation, and a discussion of his work follows.

Conceptual Models as Advance Organizers

Richard Mayer among others has taken the concept of advance organizer and elaborated upon it in his work with novice computer programmers. From the early 1970's onwards, he has conducted numerous studies using concrete and graphic models of various computer functions to determine under what conditions and with what relationship if any such advance organizers facilitate learning and retention. Like Ausubel, he maintains that meaningful learning is the process of connecting new material to existing knowledge (Mayer, 1981). Existing knowledge takes the form of schema and the process of connecting new information to it is the process of assimilation. Mayer admits that at present there is no consensus on how or what mechanisms are involved in assimilation.

Mayer identifies three processes or steps in meaningful learning related to his assimilation theory:

1) Reception: the learner attends to information so that it enters short term memory;
2) Availability: the learner must possess in long term memory the appropriate anchoring structures;
3) Activation. the learner must actively use this anchoring knowledge during learning in order that new information be connected with it.

Mayer's assimilation theory is logically an extention of that of Ausubel's, in that existing knowledge in long term memory acts as a subsumer for the assimilation of new information. The idea, Mayer suggests, is to use techniques that activate the appropriate anchoring ideas. He suggests two such techniques:

1) Providing a familiar concrete model of the system;
2) Encouraging learners to put technical information into their own words.

The present discussion will focus specifically on the first of these techniques.

Mayer states that concrete models present a framework that learners can use to incorporate new information when they lack domain-specific knowledge. This review has already discussed aspects of concrete and conceptual models related to the teaching of programming languages. The notional or virtual machine (Du Boulay et al, 1981) is such a model. Mayer has taken the concept of conceptual model and developed it within the framework of advance organizer theory. He has directly tested in numerous studies his contention that models presented prior to instruction are more effective in terms of conceptual learning and far transfer than models presented after instruction.

Mayer enumerates the results of research into the use of manipulatives (eg coins and blocks) to teach elementary mathematics in order to make computational procedures more concrete (Weaver and Suydam, 1972, Resnick and Ford, 1980), and of titles, models, and advance organizers to increase recall of technical text (Bransford, 1972; Dooling and

Lachman, 1971; Dooling and Mullet, 1973). The studies show the efficacy of advance organizers as opposed to post organizers presented after instruction.

Several studies conducted by Mayer himself (Mayer, 1979a, 1979b) show that advance organizers are most useful in situations where learners do not possess prerequisite concepts (eg. for technical and unfamiliar material), for low ability students, and for inexperienced students.

In contrast to Ausubel (1968), who claims that organizers have more effect for factual concrete information, Mayer (1981) claims that he has found consistent evidence that they have a stronger effect for unfamiliar abstract material. In addition, contrary to Hartley and Davies' (1978) position that advance organizers prove more effective with high ability learners, Mayer claims that low ability learners benefit more. Such contradictory conclusions are not uncommon in the literature, and point to differences in research and design methodologies and standards, as well as to inconsistencies in operational definitions of advance prose and graphic organizers.

Mayer's studies manipulate concrete and graphic organizers in terms of position within the treatment, retention interval, and category of learning and retention facilitated by the treatment. In one study (Mayer, 1975), learners were given a concrete graphic model of various computer functions (eg. input; output; memory; executive control) before they read a textual description of these functions describing a BASIC-like language with a restricted syntax. A control group received only the manual. The model provided concrete analogies for four major functional units.

1) Input was represented as a ticket window at which data was waiting to be processed;
2) Output was represented as a message note pad;
3) Memory was represented as an erasable scoreboard in which there was natural destructive read-in and non-destructive read-out,
4) Executive control was represented as a recipe or shopping list with a pointer to indicate the line being executed.

A ten-page manual, given to all students, described seven BASIC-like statements (READ, WRITE, EQUALS, CALCULATE, GOTO, IF, STOP) All subjects were given a test consisting of six problem types involving simple generate-statement and generate-non-loop problems, as well as more complex problems requiring some learning transfer and interpretation Mayer claims that on problems requiring some transfer, the model group excelled becuase the model provided an assimilative context in which novices could relate new technical information to familiar concrete analogies On problems requiring simple retention of presented material, the model had no effect, and Mayer states that this is consistent with earlier results in other domains in which models enhance transfer performance but not simple retention (Scandura and

Wells, 1967; Groteluueshen and Sjogren, 1968; Mayer, 1977).

A later study (Mayer, 1976) used similar materials but manipulated the position of the model, one group receiving it before and another group receiving it after reading the textual materials. The locus of effect of models, according to Mayer's assimilation theory, is located prior to learning because they provide a meaningful context within which new material may be assimilated. As predicted, the group who received the concrete model before reading the text fared better on problems requiring far transfer to novel situations, whereas the after group excelled on retention problems.

Mayer and Bromage (1980) have also tested for differential recall of idea units from text for groups receiving a model similar to that in his earlier studies, either before or after reading the text. There were three categories of idea units in the text

1) Conceptual idea units related to internal operations of the computer;

2) Technical idea units that gave examples of code;

3 Format idea units that gave rules of grammer.

Results showed that the advance organizer group recalled more conceptual idea units while the after group recalled more technical and format idea units. Mayer and Bromage claim that this pattern is consistent with the idea that good recall requires recognition of specific code whereas good transfer requires understanding of conceptual ideas.

They emphasize that if their study had evaluated learning outcomes solely in terms of amount recalled, there would have been no justification for the advantage of assimilation encoding theory over other theories of learning. However, an analysis of the type of knowledge acquired both by the advance and post organizer groups revealed significant Treatment X Performance interactions, in which the advance organizer group excelled on tasks involving transfer to conceptually more distant tasks (eg. interpreting what a program would do), and the post organizer group excelled on near transfer tasks the content of which was similar to that presented in instruction itself. The authors explain this interaction in terms of Mayer's assimilation theory. They contend that the advance organizer group benefited from a broader learning outcome in which new material was better integrated into existing cognitive structure, whereas the learning outcome for the post organizer group was more narrow.

Mayer shows that the pattern of results in his studies consistently favours low ability students. He suggests that high ability learners already possess their own models for thinking about how computers operate, and that providing an externally imposed model not only does not lead to increased learning but might actually interfere with the high ability learner's performance.

Other studies (Mayer, 1975, 1976) compared a diagrammatic model of computer operations against a flowchart of the same operations. The diagrammatic model relied on a set of meaningful past experiences using the images of scoreboards, ticket windows, and shopping lists to represent specific computer operations. Results showed that the model group excelled on posttest items requiring extention of material to novel situations in which some sort of interpretation was involved.

Mayer (1975) distinguishes between two structural variables affecting the acquisition of new knowledge:

1) External connections, which Ausubel refers to as meaningful learning set, comprise the links between new knowledge and already existing cognitive structure;
2) Internal connections refer to the link between one aspect of new information and another aspect of new information. Ausubel calls this rote learning.

Mayer (1976) contends that in the diagrammatic model groups, the learners acquired cognitive structure with strong external connections but weak internal ones. In other words, they tended toward meaningful learning as defined by Ausubel. The situation was reversed for the flowchart group, who tended toward the rote learning of the computer operations represented in the flowchart. Similarly, in the Mayer and Bromage (1980) studies of differential recall patterns for advance and post organizers, it is argued that the advance and not the post organizers encouraged the development of strong external connections to existing cognitive structure, and hence, meaningful learning.

Mayer concludes that prior exposure to a familiar model especially to a concrete model that can be related to new information is a powerful learning aid.

In this respect, Royer and Cable (1975, 1976) report on two studies they undertook into the facilitative effects of illustrations and analogies presented prior to the learning of more abstract material. Students were presented with either an initial concrete description or an abstract description supplemented with illustrations and concrete analogies of concepts related to heat flow and electrical conductivity in metals. The authors hypothesized that this initial passage would provide a "knowledge bridge" between information already existing in the learner's cognitive structure and new information contained within a second target passage. (Note: It is interesting to observe that while such a hypothesis is strikingly similar to both Ausubel's subsumption and Mayer's assimilation theories, Royer and Cable (1975, 1976) make no attempt to situate their work within this broader context).

The studies presented in concrete terms the internal structure of metals by drawing analogies between metallic structure and familiar objects. For example, the regular crystalline structure of metal was represented in terms of a tinker toy model where the discs in the model represented the molecules in the metal and the sticks between the discs represented the chemical bonds between them. Students presented with such concrete and illustrative models recalled significantly more idea units from the second abstract passage than either a control group or a group who had read an initial abstract passage. The authors conclude that such instructional strategies provide a context for comprehending difficult or abstract material when such material cannot be readily assimilated into existing cognitive structure.

The conclusion of Royer and Cable (1975, 1976) that students presented with concrete and illustrative models recalled significantly more idea units from a second abstract passage, is consistent with the research findings of Mayer (1975, 1976) and Mayer and Bromage (1980). Such consistency, while not conclusive, does argue for further investigation and subsequent application of conceptual and concrete models as advance organizers in the learning of difficult or abstract technical information.

Graphics and Text in CAL

Research into the application of graphics and text in CAL is still in its infancy and the literature reflects this fact One review (Moore and Nawrocki, 1978) found that research directed towards the effective use of graphics in CAL was virtually non-existent. The review concluded that there is very little empirical evidence to support the use of graphics in any instructional environment. It suggests that future research should try to determine the effectiveness of computer graphics as a function of the type of graphics, subject matter, task, and learner characteristics.

The concept of graphic or textual adjuncts to instruction is not novel The quantity of research that has focused on the instructional effect of illustrative adjuncts to textual material is truly formidable. The review by Levie and Lentz (1982) concluded that the learning of information presented both in text and illustrations will be facilitated, and that illustrations can help the learner both understand and recall what she reads.

Dwyer (1978) states that, "The use of certain types of visual illustrations to complement self-paced instruction can significantly improve student achievement of specific educational objectives" (p 129) But he goes on to caution that, "For certain types of educational objectives, printed instructions without visualization is as effective as visually-complemented instruction" (p 132) The degree to which an educational objective is aided by illustrations is a function of the emphasis given to knowledge about spatial information in the test of learning The Levie and Lentz (1982) review concludes that illustrations facilitate the learning of text-redundant information but have little effect on the learning of text information that is not illustrated

Research has also studied the use of maps, diagrams, learner-produced drawings and graphic organizers Whereas maps show spatial relationships, diagrams and graphic organizers show conceptual relationships The research concluded that learning was facilitated when the non-pictorial illustrations were related to the textual material (Bartram, 1980, Schwartz and Kulhavy, 1981)

Sless (1981) states that diagrams and graphs are, "techniques for transforming or transposing information from one form which is either impossible or difficult to manipulate or operate upon, into another form which makes these operations possible or easier" (p 144) He argues that a graphic or graphics present information which is more intelligable than it would be if presented by any other communication or symbol system In this sense, it is a form of communication which structures ideas in terms of providing a model with which characteristics may be classified

MacDonald-Ross (1977) argues that the function of graphics is to display conceptual information whether it be numerical, logical, spatial, or temporal He suggests that the invasion of cognitive psychology by ideas from artificial intelligence is significant for a theory of graphic communication in that the graphic format is a representation for problems of a certain class Simon (1969) has stated that, "one of the key steps in solving a problem is to represent it so as to make the solution transparent" Loewe (1971) has argued along similar lines that the transparency of the problem situation directly affects the possibility of insightful learning. In this sense, graphics allow people to manage certain tasks that would be impossible or difficult if represented verbally or in text alone.

There are numerous functional classifications of such illustrative materials as diagrams, maps, charts and other non-representational drawings Duchastel (1978) proposès a three-part classification·

1) Attentional-creates interest and motivation;
2) Explicative-explains concepts not expressable in words;
3) Retentional-increases retention and recall of concepts.

The retentional function of illustrations is based on Paivio (1971) who argues that human memory of pictures is less degradable than that for verbal information.

MacDonald-Ross (1977) offers a four-part classification:

1) Iconic-shows what objects look like and labels parts;
2) Data display-displays results of empirical observation;
3) Explanatory-shows logical relationships between ideas;

4) Operational-helps user perform specified tasks

MacDonald-Ross agrees with Duchastel (1978) that the functions of illustrations can, and often do, overlap

Levin (1981) proposes a seven-function classification which includes similar categories to those of both Duchastel and MacDonald-Ross. While other classifications have been proposed (Levie and Lentz, 1982), it is the commonalities and not the differences between them which stand out These commonalities include

1) Motivation-illustrations create interest and draw attention to material;
2) Explication-illustrations explain or show relationships between concepts which would be otherwise difficult or impossible to communicate;
3) Retention-illustrations help in the retention and recall of conceptual and factual information.

Diagrams depict the organization and structure of the key concepts in a content area. In this sense, they are logical pictures. Winn and Holliday (1981) in a review of research on learning from diagrams, conclude that diagrams help by showing, "which concepts go with which others, aiding generalization and discrimination by replacing critical verbal information with graphic devices such as lines and arrows" (p 22) Holliday (1976) found that learning improved 30% when science students were given a relevant diagram. In another study, however, students did better when presented with a flow diagram only rather than with the diagram and text components (Holliday, 1976b). High school students were presented with block word and picture word diagrams depicting the carbon dioxide, oxygen, water and nitrogen cycles. Some students received a textual version in addition to the diagrams.

Winn (1980) found that block word diagrams facilitated the organization of content more accurately than did text alone for high verbal learners. There was no difference for low verbal learners, amd Winn (1981) suggests that materials used in science curricula should be designed in terms of the abilities of the specific population using them. He emphasizes that both pictures and diagrams can facilitate the learning of discrimination and generalization skills in addition to showing representations of concepts realistically and the relationships between them. Winn (1982) suggests as well that diagrams can show the sequence of concepts in terms of step-by-step processes such as food chains and insect metamorphosis. He concludes that the effective use of diagrams must take into account the functional aspects of pictorial elements, conceptual grouping through layout, and conceptual sequencing.

Graphic organizers are schematic representations of the relationships between concepts. In addition, they are a type of advance organizer in terms of activating and providing relevant prior knowledge. This review has already discussed the inconclusive research findings concerning the efficacy of advance organizers. One critical review of advance organizer studies (Barnes and Clawson, 1975) suggested that perhaps more research should address the area of non-textual organizers or graphic organizers

In this sense, Bernard et al (1981) found that both an image and a verbal organizer facilitated learning of a short text passage compared to a no-organizer control. The authors conclude that contextual retentional images and their verbal counterparts are particularly useful when learners do not possess prior knowledge in the subject area.

Two reviews (Moore and Readence, 1979; Barron, 1980) concluded that results of graphic organizer studies are small or mixed at best.

Illustrations facilitate learning by improving both the comprehension and retention of material (Levie and Lentz, 1982). In addition, illustrations can provide information which is not or cannot be included in text.

In terms of graphics within a CAI environment, Rigney and Lutz (1976) investigated the effects of presenting concepts related to electrochemistry via a system of interactive animated computer graphics. Graphic analogies were designed to supplement the verbal description of concepts related to a simple cell or battery. Each of two groups of undergraduate students received either a textual description with graphic adjuncts or simply the textual description with textual elaboration supposedly paralleling the graphic component Results showed both better recall and recognition test scores in addition to more positive student attitudes for the graphic group compared to the group that had received only textual materials

A more recent study (Alesandrini and Rigney, 1981) involved similar instructional content and lesson strategy, but added a pictorial review component in which learners actively interacted with a graphic display via a touch-sensitive screen. Their objective was to create a model of a working battery by rearranging images of the battery components illustrated on the screen. The authors state that such an exercise allowed the learner to practice all of the concepts taught during the lesson itself A control group engaged in 30 minutes of game playing against the computer.

Results indicated that learners in the pictorial lesson/review group recognized significantly more items on the pictorial test than those who had been exposed to a verbal lesson and no review. However, as the authors admit, when time-on-task is taken into account the advantages of such an instructional strategy become less than clear.

Though there is a lack of research into the instructional effects of graphics in CAL, several studies have investigated the differential effects of alternative levels of graphic displays. King (1975) studied the effects of computer graphics in the teaching of the sine-ratio concept. He found no significant differences between groups receiving

1) CAI text only,
2) CAI text and still graphics;
3) CAI text and animated graphics

He concludes that perhaps the learning task was not difficult enough to be sensitive to the effects of the graphics used.

Moore et al (1979) compared the effects of three levels of graphic displays:

1) Low level: boxed alphanumerics and schematics;
2) Medium level. line drawings;
3) High level: line drawings and animation.

There was no significant difference found between groups exposed to the different display levels in terms of retention or lesson completion time. Moore et al (1979) conclude that addition of more realistic or sophisticated graphics does not insure an increase in learning. They add that the question which should be asked about the role of graphics in CAI is not one concerning the degree of complexity involved, but rather the question of what are the appropriate circumstances in which to use graphics. They stress that CAI's greatest potential lies in its ability to individualize instruction using interactive techniques. In this sense, the use of interactive graphics, they conclude, warrants further investigation.

McKenzie et al (1978) agree that it is in the interactive capability of the computer especially as this relates to graphics where CAL can have the greatest potential impact in terms of individualizing instruction They concede that the use of graphics in education is not new. However, the combination of graphic capability and interaction, they emphasize, makes for a powerful learning tool because the student is forced into an active mode. The authors state that it is active interaction and not passive reception that leads to effective reinforcement of learned material They suggest that a large portion of the science curriculum consists of a set of inter-dependent laws, describing functional relationships between observable and derived quantities. Such relationships, they conclude, are more easily understood in graphical than analytical form and, therefore, the use of interactive graphics can maximize understanding Topics such as symmetry, waves and statistical distributions because of their intrinsically graphic nature are naturally given to such graphic presentation.

While much of the experimental evidence does not support the use of graphics in CAL, there are staunch advocates of graphics-oriented CAL environments

In this sense, the effective use of computer graphics in education has been championed if somewhat nonexperimentally by Alfred Bork at the Educational Technology Center of the University of California Bork has argued repeatedly for increased application of computers in education because, he states, " The computer is the first technological innovation that enables us to move back to a more Socraticlike environment in which the teacher can respond fully to each student" (Bork, 1981).

Bork identifies two principle aspects of graphics in learning environments, both of which have been noted previously:

1) As a conveyer of information that might not have been or could not have been conveyed in a non-iconic medium;
2) As a motivational tool to assist in orientating students towards learning.

He does not emphasize any theoretical base for his recommendations, and one gets the impression that he is speaking on an intuitive level, though one tempered by a fair degree of experience. Bork has expressed reservations concerning the application of the scientific experimental paradigm to educational research (Bork, 1977). He adds that there are no simple rules concerning the use of pictures in learning and that a good teacher's intuition and experience in specific subject domains are crucial inputs into the instructional development process.

Bork emphasizes the particular quality of graphics to display information that is inherently pictorial. He gives as an example a program called MOTION which illustrates the properties of planets orbiting a sun (Bork, 1981). In addition, he states that graphics can help students build intuition in physical situations such as in calculations of velocities in space. By plotting various aspects of such systems, the student is provided with a range of experiences and develops intuitions about how systems behave. Bork (1977) champions the use of graphics in the development of problem solving skills. Interactive computing, he stresses, can provide students with insight into how problems are practically solved by showing them how to organize an attack on the problem. Graphic visualization of a path through a problem can sometimes prevent the student from wandering into areas which are not germane to the problem at hand. Bork encourages his students to represent problem situations diagrammatically in terms of generating a map or series of maps of possible processes and outcomes.

Bork does not ignore the fact that graphics in CAL can support and augment textual material. The presentation of information in both iconic and textual forms is desirable he says (Bork, undated). Unlike the printed book, graphics and text can be interwoven in time to reinforce and explicate relevant material. Graphics and text can evolve over a period of time to aid the student in understanding dynamic phenomena.

In addition, he emphasizes the use of graphics in presenting students with an overall view or organizer of a particular learning area (Bork, 1981). In this sense, he is arguing for the application of effective graphic organizers presented interactively via computer. This is a new area and one in which much fruitful research could perhaps be conducted. He has used block diagrams to illustrate major content areas. For example, in the program SPACE, a map shows the pedagogical organization to the student providing him with a menu facility in terms of concepts, advice, problems and a free-play component.

Bork sees textual layout as an important graphic variable in terms of optimizing readability, memorability and motivation (Bork, unpublished). He suggests numerous guidelines for textual layout both in terms of spatial and temporal variables. He urges the use of large areas of blank space, short lines, little hyphenaton, and natural phrase parsing. He suggests placing pauses after critical words and the use of blinking to emphasize portions of text. He reminds the reader that motivation can be increased by varying the style of the display in terms of size of type, font, and alignment. Bork emphasizes over and above these recommendations that both graphic and textual material can and should be controlled by the student in order that she feel comfortable with the development of the lesson.

This section has concluded that research into the effective use of graphics within a CAI or CAL environment is almost non-existant, and that what research does exist, promises little significant benefit from its implementation. However, due to the meagre body of current research, it would be pre-mature to come to any definitive conclusions at this point. Further research is certainly called for into the appropriate and effective use of graphics in CAI or CAL environments.

Conclusion

This review has addressed a broad range of topics related to the focal concerns of the teaching of computer programming skills, and their subsequent use in problem solving activity. It opened by reviewing research directed towards the problems that novices encounter in the learning of a first computer language in terms of planning, coding, and debugging computer programs. This was followed by a discussion of learning outcome as a function of teaching/learning strategy, in which research concluded that operation learners (Pask, 1976) tend to develop to a greater extent than comprehension learners, the macrostructures of a language which subsequently permit them to more successfully assemble language structures into effective algorithms (Coombs et al, 1982).

Computer programming was viewed as a class of problem solving activity by several researchers (Hoc, 1977; Papert, 1980; Green, 1980; Brooks, 1983; Kelman et al, 1983). The work of Card, Moran, and Newell (1983) suggesting that most cognitive activities develop from problem solving behaviour was seen as supportative of the view that the teaching of programming to novices entails a problem solving approach. The programming language, Logo, in particular, was examined in this light. The assertion that programming in Logo leads to transfer of problem solving abilities was found to be a contentious one. Research methodology in Logo problem solving was criticized for its lack of experimental precision and control (Krasnor and Mitterer, undated), and some convincing argument against the ability of Papert's unstructured "piagetian" environments to foster transfer of general problem solving ability was presented (Léron, undated)

The use of conceptual models to simplify and illuminate computer systems and languages was determined to be an effective instructional strategy by many researchers (Miller, 1974, Mayer, 1975, 1976, 1979, 1981, Sime et al, 1977, Myers, 1980; Young, 1981; Moran, 1981a, 1981b, Howe and Ross, 1981, Du Boulay et al, 1981). The literature on advance organizers was less conclusive but research into the application of conceptual and concrete models as advance organizers appeared to yield positive results in terms of far transfer of conceptual knowledge and skills to novel situations (Mayer, 1975, 1976, 1979, 1981; Royer and Cable, 1975, 1976, Mayer and Bromage, 1980).

Finally, research into graphics and text in CAL was inconclusive, primarily because of the small number of studies directly addressing this area. The studies that are extant argue for a less than significant effect for graphics in CAL or CAI environments. It was concluded, however, that further research is required before strong conclusions can be drawn.

CHAPTER III

Hypotheses

The objective of this thesis was to determine the effectiveness of an instructional strategy for teaching abstract programming concepts to novice programmers The strategy involves the use of models presented graphically on a microcomputer VDT screen, either immediately before or immediately after a basic CAL lesson on word and list processing The review of literature has presented research supportative of the use of conceptual and concrete models as aids towards the teaching of computing concepts (Mayer, 1975, 1976, 1979a, 1981; Mayer and Bromage, 1980, Du Boulay and O'Shea, 1978, 1981; Du Boulay and Howe, 1981; Du Boulay et al, 1981) The position of these models has also been investigated, and research supportative of their use as advance organizers has been cited (Mayer, 1975, 1976, 1979a, 1981; Mayer and Bromage, 1980, Royer and Cable, 1975, 1976).

The majority of this research has utilized either concrete physical models or graphic/textual models printed on paper. This thesis attempted to exploit the virtually untested instructional potential of animated computer graphics in presenting a series of graphic models specific to Logo word and list operations. While the research into the instructional benefits of both computer graphics (Moore and Nawrocki, 1978) and graphic organizers (Moore and Readence, 1979) has been either non-existent or disappointing, there has been virtually no research into the application of models presented graphically via the microcomputer This thesis attempted such an application.

Research has established the efficacy of utilizing advance organizers and models under certain circumstances and with certain learners, specifically, for presenting new, technical information to lower-ability students (Mayer, 1979b) In addition, studies have concluded that illustrations, including graphics, diagrams, and maps do facilitate both comprehension and retention (Levie and Lentz, 1982). The author undertook, therefore, to exploit the facilitative effects of organizers, models, and graphics in the testing of several specific hypotheses.

1) The use of models presented graphically either before or after a CAL lesson will lead to greater comprehension and retention of lesson content, compared to a CAL lesson with no models.

2) The use of models presented graphically either before or after a CAL lesson will lead to greater transfer of learning to novel problem solving situations, compared to a CAL lesson with no models.

3) The use of models presented graphically as an advance organizer prior to a CAL lesson will lead to increased comprehension and retention of lesson content, and to increased transfer of learning to novel problem solving situations, compared to either a no-model or a post-lesson model instructional strategy.

Rationale for Hypotheses

The rationale for Hypotheses 1 and 2 may be found in the research cited previously involving concrete and conceptual models presented as part of the overall instructional strategy (Du Boulay and O'Shea, 1978, 1981; Du Boulay and Howe, 1981; Du Boulay et al, 1981; Howe and Ross, 1981). These researchers believe that providing a simplified model of the computer, variously termed a virtual or notional machine, can make abstract programming concepts more concrete and meaningful, and hence lead to increased comprehension and retention. The use of models to externalize the otherwise hidden processes involved in many programming operations is thus seen as a potentially effective instructional strategy for a population of novice computer users.

The rationale for Hypothesis 3 is derived from the work of Ausubel (1978), Mayer (1975, 1976, 1977, 1979a, 1979b, 1979c, 1981), Mayer and Brommage (1980), and Royer and Cable (1975, 1976). These researchers have combined the concept of concrete or conceptual model with that of the advance organizer. Their research indicates that the use of models as advance organizers does have a facilitative effect on the learning and retention of abstract or technical information. Mayer (1981) has emphasized from Greeno (1980) and Simon (1980) that novice programmers lack domain-specific knowledge. One technique, he argues, for improving novices' understanding of technical or abstract material is to provide them with a domain-specific framework or model that can be used to assimilate new information. The presentation of the concrete models prior to instruction is seen as one way of providing this domain-specific framework.

Mayer (1981) argues further that the use of concrete models as advance organizers facilitates the transfer of learning to novel problem solving situations, compared to instruction with no models or to a strategy using models after instruction. Hypothesis 3 directly addresses this contention within a CAL environment.

Definitions

For the purposes of the present study, the following definitions apply:

Comprehension

Comprehension includes those objectives, behaviours, or responses which represent an understanding of the literal message contained in a communication, without necessarily relating it to other material or seeing its fullest implications (Bloom et al, 1956).

Retention

Retention involves the relative permanance of what is learned when tested in situations essentially duplicating those of the original learning (Hilgard and Bower, 1966).

Transfer of Learning

Transfer of learning implies the use of learned information in problem solving tasks which are different from what was explicitly taught (Mayer, 1981).

Problem solving

Problem solving is a process by which the learner discovers a combination of previously learned rules which can be applied to achieve a solution for a novel situation. This process produces higher-order rules which allow the learner to solve problems of a similar type (Gàgne, 1977).

Advance organizer

An advance organizer is a deliberately prepared set of ideas presented at a higher level of abstraction, generality, or inclusiveness to the learner in advance of meaningful learning in order to provide relevant anchoring concepts to which new less inclusive material may be subsumed (Ausubel and Robinson, 1969).

Conceptual model

A conceptual model is a definite representation or metaphor which guides a learner's actions and helps him/her to interpret a system's behaviour (Young, 1981).

Graphic model

A graphic model is a schematic representation of a concrete or conceptual model, using lines, planes, and animation.

Novice programmer

A novice programmer is a naive user or non-expert in computer technology who has little or no experience in computer programming and who, therefore, lacks domain-specific knowledge (Mayer, 1981).

CHAPTER IV

Method

Population and Sample

The population for the study comprised graduate and undergraduate students from the Education Departments of Concordia and McGill Universities. They were novice programmers in the sense previously defined, that is they were naive users or non-experts in computer technology with little or no previous programming experience. They were typically students completing an undergraduate degree in education, a Diploma in Computer Assisted Learning, or a Masters Degree in Educational Technology.

Subjects were drawn from a pool of three classes, two from Concordia University and one from McGill University. All three classes were introductory in nature. Course content included the study of computer literacy, evaluation of software, and simple programming (BASIC and Logo turtlegraphics). Subjects had not received any instruction in word and list processing.

The sample was selected from the three classes on a volunteer basis. Students were randomly assigned to one of two treatment groups (Group A and Group B), or to a control group (Group C). Group size varied between 11 and 15 subjects, for a total of 41 subjects.

Design

A 3 X 2 factorial design was used, with one between-group factor (Instructional Strategy) and one within-group factor (Retention Interval). Two evaluation instruments were administered, an immediate and a delayed (one week) posttest The two posttests contained parallel but not identical items. All three groups received both posttests. In addition, all groups received the same basic content in a CAL lesson delivered via a network of IBM Personal Computers located in Concordia University's Computer Center.

All groups received an aptitude test in the form of a formal operations measure (FOM, Tomlinson-Keasy and Campbell, undated). This measure, which was used as a covariate in the analysis of data, was included in order to allow a more focused set of conclusions to be drawn in terms of any differential learning effects between high and low ability students. Mayer (1981) claims that it is low ability students who benefit most from concrete and graphic models because higher-ablility students already possess their own sets of models. Hartley and Davies (1978) on the other hand believe that high ability students can derive the greater benefit because they are more capable of interpreting and using the models. FOM was used in this study because formal operations are assumed to underly comprehension and transfer in mathematically and logically related activities such as word and list processing.

Treatment Group A received, in addition to the basic content, a series of models presented graphically in a CAL package as an advance organizer, immediately before the basic lesson. This series of models paralleled the content presented within the basic CAL lesson itself

Treatment Group B received, in addition to the basic content, a series of models presented graphically in a CAL package immediately after the presentation of the basic lesson. This series of models was identical to that received by Treatment Group A with the difference being that it was presented after and not prior to the basic lesson.

The control group, (Group C), received the basic CAL lesson only. A forced review of the content in this lesson was included to equalize time-on-task between the three groups, and to minimize the possible effects of repetition on performance in terms of Groups A and B.

Procedure

Subjects were informed by the author that they were participating in a study attempting to determine effective instructional strategies for the teaching of programming concepts. Each of the three classes involved in the study arrived at the testing room on separate days. Due to experimental constraints, randomization was not based on covariate scores. Instead, randomization was effected by assigning individual subjects to one of the three groups. Subject 1 was assigned to Group A, subject 2 to Group B and so on. Subjects were run through their respective treatments in pairs, each pair at an IBM Personal Computer equipped with a keyboard for entry and a colour graphics monitor. There was no time limit and students were provided with a menu at the end of the lesson from which they could review a part or all of the lesson content. Average time spent at the computer was 30 minutes.

Directly after finishing the lesson, each subject was administered the immediate posttest, consisting of 50 multiple choice and short answer items. Again, no time limit was imposed, and the average time spent completing the posttest was 35 minutes. After completing the posttest, each subject was excused from the testing room. One week later, the subjects were administered the delayed posttest in their respective classrooms. The author was not present during this period.

Materials

Basic CAL Lesson

The basic lesson comprised a CAL package covering part of the word and list processing subset of the programming language Logo. The lesson itself, was developed following the basic principles of instructional design (Dick and Carey, 1978) and programmed in IBM-LCSI Logo.

Logo uses operations or procedures to manipulate words and lists. Words and lists are called objects. The basic lesson introduced eight Logo operations, defined their functions, and provided examples of their application to real words and lists. Students were asked to respond to questions by typing at the keyboard what they believed the effect of each Logo operation had on specific words and lists.

The eight Logo operations were: FIRST
LAST
BUTFIRST
BUTLAST
FPUT
LPUT
WORD
SENTENCE

Each of these operations takes an input and outputs a specific Logo object (word or list). For example, the operation FIRST takes a word or list as input and outputs the first character of the word, or the first word of the list. The operation LPUT takes an object and a list as input and outputs a new list formed by putting the object at the end of the list. The operation SENTENCE takes any number of words or lists as input and outputs a single list made up of this input.

The presentation of each operation occupied approximately two to three screens. A functional definition was provided followed by several examples of the effect of the operation on various inputs. The examples focused upon individual lines of code rather than complete Logo procedures (programs) because it was the specific effect of the operation on its input that the lesson was attempting to exemplify, and not a broader explanation of the Logo language.

The eight operations included in the lesson were selected from a much broader suite of operations and commands contained within the Logo language. The operations selected for the study, however, had one distinguishing feature: they all transformed specific input into specific output in a manner that could be readily modeled in a graphic manner. Because they all transformed words and lists into new words and lists, the author felt that these operations were good candidates for the study. Logo operations not included in the study, such as COUNT, do not necessarily transform words and lists, but perform other, related tasks (COUNT outputs the number of elements in a list).

Graphic Models

The series of models presented in the CAL packages, was designed to simulate graphically what Logo is doing in transforming input to output for each of the eight operations previously enumerated. The models, also programmed in Logo, visually reproduced on the screen the operations performed by Logo on words and lists during such operations as searching through a word or list for a particular character or word, concatenation, testing of words and lists against other words and lists, and outputting specific parts of words and lists.

The student was able to see how any operation altered a given word, list, or list of words and lists. These operations were modeled using strings of boxcars, forming "word trains" capable of being manipulated by a switching yard of Logo operations. The concept that trains are manipulated in freight yards is one that is presumed to be familiar to the population, and was thus seen as an appropriate concrete model (presented graphically) of Logo word and list operations.

The student actively used the models by inputting his/her own words and lists which appeared on the VDT monitor, each character or word carried within its own boxcar. Such a "train" of boxcars was then subjected to the specific Logo operation being currently modeled as the train passed through a "switching yard". A new train emerged from the yard, altered by the specific operation exercised upon it.

For example (see Appendix B), the graphic model of the operation BUTFIRST (BUTFIRST takes a word or list as input and outputs a new word or list minus the first character or word), showed a train of characters or words (eg. THIS IS A LIST) in the upper portion of the monitor. The student was asked to activate the BUTFIRST operation via the keyboard, and a new train abbreviated by the first word appeared in the lower portion of the screen (eg. IS A LIST). In terms of its use as an instructional resource, the series of graphic models could be interpreted as a graphic organizer available to the student first learning Logo word and list processing before, during, and after the more abstract presentation of the language. In a sense, it could be considered a graphic simulation of abstract processes, allowing students to "play" graphically in a more concrete environment. The student would be permitted to experiment with varying inputs acted upon by specific individual operations or combinations of operations, resulting in immediately verifiable output. Such a simulation would provide the student with feedback of specific operations on specific inputs. In creating an environment in which the student is able to experiment at his/her leisure, the graphic simulation might encourage active participation in an area of programming that can be difficult for some students to visualize and understand.

For the present study, however, the application of the series of graphic models was more strictly controlled and manipulated in terms of the experimental design previously described.

Both the basic lesson and the graphic models underwent a series of tryout/revision cycles recommended by Nathenson and Henderson (1980) for the formative evaluation of novel instructional materials. Students selected from the population were run on an individual basis through the material, revisions were then made based on these tryouts, after which additional tryouts with different students were undertaken. Four subjects participated in this tryout/revision evaluation.

The major changes to the material during this process of evaluation involved the clarification of definitions and question items, the re-arrangement of several screens in terms of text and graphics, and the elimination of certain distracting elements within the programs (blinking cursor, inappropriate use of colour, distracting sound effects). In addition, a suite of sub-routines was written to augment the error-handling capabilities of the programming itself, in order that the programs would not be interrupted by unanticipated student input. Sub-routines were written to protect the programs against the possibility of students pressing the return key more than once, pressing the return key with no input, and inputting sentences and words whose length could not be handled because of graphic limitations.

The programs were written in IBM PC Logo, Version 1.0, using a structured, block by block approach. In this manner, programming time was drastically reduced by virtue of the fact that each of the three programs required was assembled from suites of previously-written program blocks, each of which performed a specific function.

Evaluation Instruments

The immediate and delayed posttests contained test items evaluating both comprehension and retention, and transfer to novel problem solving situations (see Appendix A). In terms of the literature on the use of models as an aid to problem solving in computer programming, transfer has been defined as the ability to solve programming problems that are different from those presented in the lesson but which can be solved using the information provided (Mayer, 1981).

Comprehension and retention problems involved the direct use of information provided within the lesson. Transfer problems, on the other hand, involved the application or interpretation of lesson content to novel situations. As previously noted, the two tests contained parallel but not identical items.

An item which evaluated comprehension and retention presented the student with a specific function (eg. Takes a list of words as input and outputs the list minus the first word) and asked for the name of the Logo operation which performs the function. Alternatively, the student was presented with two lists of words, and asked to identify the Logo operation which was used to transform the first list into the second.

Items which evaluated transfer to novel problem solving situations presented problems which had not been directly covered within the instructional material, although the student would have received all the information necessary in order to solve them. Several transfer items presented the student with two lists of words, and asked him to identify the specific combination of Logo operations which was used to transform the first list into the second. For example, the student was presented with these two lists: (This is a real puzzler) and (is a real), and asked to identify the combination of operations which took the first list as input and output the second.

Alternatively, several transfer items presented the student with a list of words and a combination of Logo operations, and required the identification of the output list. For example, the student might have been presented with this list: (This is the end) and this combination of operations: BUTLAST BUTLAST, and be asked for the new list resulting from the application of the operations.

The items in both posttests were classified according to whether they tested for comprehension and retention or transfer to novel problem solving situations. This classification was intended to be used in the analysis of data in an attempt to identify any resulting Treatment X Learning Outcome interactions.

The testing instruments were also formatively evaluated in conjunction with the lesson materials. Several cycles of student tryout/revision led to major changes in both the number and the difficulty of the items. Initial tryout revealed that several items were either too difficult or too easy, and these were replaced by other items. In addition, the weighting of the tests was altered so that items testing for transfer of learning were more prominent. This was done in an attempt to make the instruments more sensitive to effects of treatment on problem solving skills.

CHAPTER V

Results

Introduction and Hypotheses

The general hypothesis of this study stated that the provision of models in the teaching of programming languages would concretize the otherwise abstract concepts involved in such operations as word and list processing, and consequently have a positive effect on both the comprehension and the ability to use a language in problem solving situations.

Specifically, three hypotheses were put forward:

1) The use of graphic models either before or after a CAL lesson would lead to greater comprehension and retention of lesson content, compared to a CAL lesson with no models;

2) The use of graphic models either before or after a CAL lesson would lead to greater transfer of learning to novel problem solving situations, compared to a CAL lesson with no models;

3) The use of graphic models presented prior to a CAL lesson as an advance organizer, would lead to both increased comprehension and retention, and increased transfer, compared to either a no-model or a post-lesson model instructional strategy.

These hypotheses divide the performance measure into two components: a comprehension/retention and a transfer component. The rational for such a division, as previously stated, is found in the literature on the teaching of programming with graphic and concrete models (Mayer, 1975, 1976, 1981), especially as this relates to the use of advance organizers. The hypothesis that the application of models as advance organizers produces a differential learning effect in favour of transfer skills, has been tested and found credible. The present study attempted to replicate prior research within a CAL envoronment.

Experiment

A two-way analysis of covariance (ANCOVA) with repeated measures was conducted as the basis for the interpretation of the test data and the determination of the validity of the hypotheses. The between-subjects factor, Instructional Strategy, contained three levels: 1) A: Advance organizer graphic model; 2) B: Post organizer graphic model, and 3) C: No graphic model. The within-subjects factor, Retention Interval, contained two levels: 1) Immediate; and 2) Delayed (one week). As discussed in the previous section, the score in the FOM test was used as the covariate.

The dependent measure, Learning Outcome, was originally divided into two variables: 1) Comprehension / Retention and 2) Transfer, in order to directly address the hypotheses. The result of a principal components analysis (Gnanadesikan, 1977) on these variables, revealed only one significant factor (accounting for 74% of the variance) to which both contributed equally (Factor Score Coefficients:

Comprehension/Retention .58; Transfer .58; Correlation Coefficient between variables: .48, p < .002). A discriminant analysis performed on the dependent variable further confirmed this conclusion, in that no significant discriminant functions were uncovered (Chi-square: 9.2026, p > .05). Based on these analyses, it was decided to combine both Comprehension/Retention and Transfer into one variable by summing each subject's scores on both. The variable thus obtained will be heretofore called the dependent variable.

As was previously mentioned, the immediate posttest consisted of 50 multiple choice and short answer items. The author concluded that this number was unnecessarily large, and consequently, an abbreviated version consisting of 25 items was extracted from the immediate posttest. Parallel items, based on this version, were written for the delayed posttest. A reliability test was run to determine both the reliabilty of the instruments themselves and any appreciable loss in reliability due to the abbreviation process. Reliability coefficients for the full and reduced immediate posttest were .90614 and .82793 respectively. While this represents a reduction, it was felt that the abbreviated version was adequate to the task. THe analysis from this point on is based upon results from these abbreviated evaluation instruments.

Pearson Correlation was used to determine the predictive power of the covariate. Table 1 shows the correlation coefficients between the FOM test results and the immediate posttest results for each of the three groups, as well as that between the FOM test and the combined group scores on the dependent measure. Significant coefficients are indicated with an asterisk. As may be observed, the significant correlation coefficients suggest that FOM may be a significant predictor of performance on the evaluation instruments used in this study.

The means and standard deviations of Groups A, B, and C on the dependent variable are summarized in Table 2. The figures contained within brackets represent the immediate posttest means and standard deviations for subjects who took both the immediate and delayed posttests. For descriptive purposes only, the combined performance measure has been presented in the two-level format originally intended, in addition to the single measure formed from the combined values. Examination of the means and standard deviations reveals a substantial difference in group means for the immediate posttest scores, with the Transfer component of these scores exercising the greater influence in this difference (Means: Group A = 34.27; Group B = 29.07; Group C = 25.18). Though a rigid statistical analysis does not permit further analysis in this direction, the high Group A mean does indicate some support for the hypothesis that graphic models presented as advance organizers do influence in a positive direction, performance on transfer of learning more than on comprehension or retention

The results of the two-way ANCOVA with repeated measures

Table 1

Correlation Coefficients Between FOM and Immediate Posttest

Groups	n	Corr. Coefficient	p
A	14	-.0204	.945
B	15	* .5146	.050
C	11	* .7176	.013
Total	40	* .4091	.009

Table 2

Posttest Group Means and Standard Deviations - Raw Scores

		Posttest (Immediate)					
Groups	n	Comp/Ret.		Transfer		Total	
		$\bar{X}$	SD	$\bar{X}$	SD	$\bar{X}$	SD
A	15 (14)	8.60 (8.57)	1.18 (1.22)	25.67 (25.14)	6.51 (6.42)	34.27 (33.71)	7.09 (7.01)
B	15 (12)	8.80 (8.83)	0.86 (0.94)	20.27 (18.92)	9.02 (9.35)	29.07 (27.75)	9.15 (9.55)
C	11 (9)	8.27 (8.22)	1.79 (1.86)	16.91 (16.89)	8.41 (9.18)	25.18 (25.11)	9.48 (10 24)

		Posttest (Delayed)					
Groups	n	Comp./Ret.		Transfer		Total	
		$\bar{X}$	SD	$\bar{X}$	SD	$\bar{X}$	SD
A	14	7.78	1.42	23.07	8.46	30.86	9.27
B	12	8.58	1.16	21.67	9.15	30.25	9.68
C	9	8.33	1.58	19.44	8.73	27.78	9.54

are presented in Table 3. As may be observed, there is a significant interaction ($F = 3.98$, $p < .03$). In order to test the assumption of homogeneity of the regression equations (of no covariate-by-factor interaction), the procedure suggested by Nie, Hull, Jenkins, Steinbrenner, and Brent (1975, pp. 381 - 383) was followed. Regression of posttest with treatment group, FOM, and treatment group by FOM yielded an R square of .41046; regression of posttest with treatment group and FOM yielded an R square of .30859. Thus,

$$F_{inter} = \frac{(R^2_{full} - R^2_{no\ inter}) / (k1 + k2 + k1k2)}{(1 - R^2_{full}) / (N - K - 1)}$$

$$= \frac{0.050935 / 2}{0.016844 / 35}$$

$$\cong 3.024$$

which is not significant at the $\alpha = .05$ level. ($F_{.05}(2, 35) > 3.23$).

A graph of adjusted group means is presented in Figure 1 which suggests a net loss in performance for Group A and a net gain for Groups B and C, between the immediate and

Table 3

ANCOVA (Repeated Measures) Summary - Results

Source	SS	df	MS	F	p
Group (G)	410.59	2	205.30	1.58	.222
1-ST Covar	831.53	1	831.53	6.40	.017
Error	4027.63	31	129.92		
Repeated (R)	10.03	1	10.03	.64	.428
R x G	123.84	2	61.92	3.98	.029
Error	498.36	32	15.57		

Table 4

Adjusted Cell Means for Dependent Variable

Group	n	Immediate Posttest	Delayed Posttest
A	14	33.96	31.11
B	12	25.65	28.15
C	9	27.52	30.19

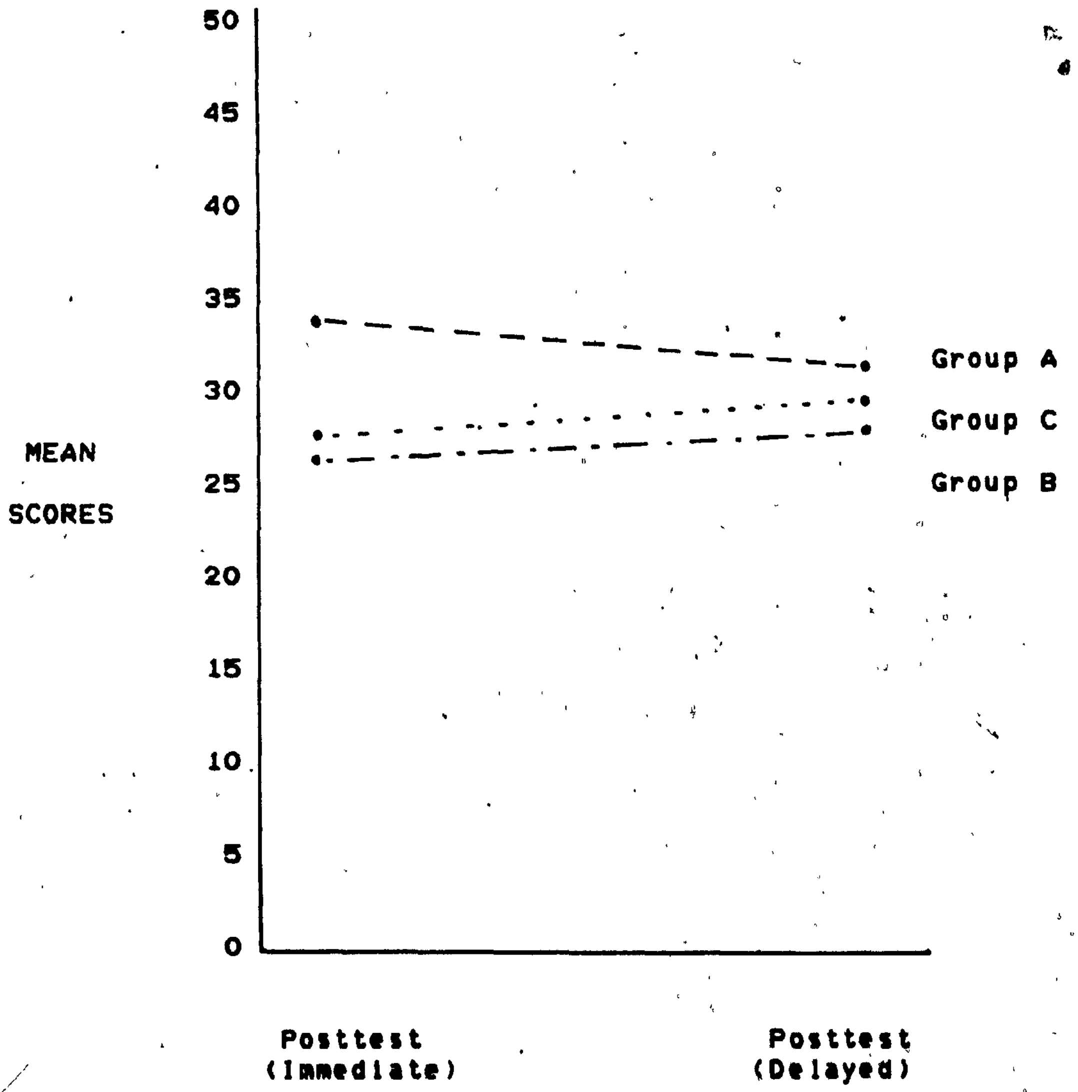


Figure 1. Comparison of adjusted posttest mean scores for Groups A, B, and C

delayed posttests. This result prompted further analytical comparisons in an attempt to determine the locus of this interaction.

Following the procedure suggested by Keppel (1973, pp 442 - 454) for analytical comparisons in mixed designs, two oneway ANOVA's were subsequently performed in order to determine simple main effects for each of the two posttests, in terms of the between group factor. Table 5 presents the results of the ANOVA for the immediate posttest, and as may be observed, there is a significant F value ($F = 3.71$, $p < .05$). Table 6 presents the results of the ANOVA for the delayed posttest, which in this case yielded a non-significant F value, ($F = .305$, $p > .05$). It was therefore concluded that there were significant group differences on the immediate posttest but none on the delayed posttest.

Pursuant to these analyses, a post hoc analysis (Keppel, 1973, pp 442 - 454) was undertaken on the immediate posttest scores in an attempt to pinpoint significant group differences. Scheffe tests revealed that Group A was significantly different ($p < .05$) from Groups B and C. Groups B and C were not significantly different. Such a result lends support to the hypothesis that graphic models do facilitate certain categories of learning when they are presented as advance organizers, prior to more traditional instruction. In spite of this, the drop in the mean for Group A over a one-week period does introduce serious doubts as to the permanence of this effect.

Table 5

ONEWAY Summary - Posttest (Immediate) - Results

Source	SS	df	MS	F	p
Between	541.28	2	270.64	3.71	.03
Within	2775.50	38	73.04		
Total	3316.78	40			

Table 6

ONEWAY Summary - Posttest (Delayed) - Results

Source	SS	df	MS	F	p
Between	54.77	2	27.38	.305	.740
Within	2875.52	32	89.86		
Total	2930.29	34			

CHAPTER VI

Discussion

The results of this study, while not entirely encouraging, do argue for a closer look at the application of graphic models within instructional strategies directed towards novice programmers. Specifically, group means in the immediate posttest were higher for the two groups (Groups A and B) that received ~~the~~ graphic model, and significantly higher for the group that received the model as an advance organizer. Such findings are supported by prior research into the application of models as advance organizers (Mayer, 1975, 1976, 1981; Royer and Cable, 1975, 1976). The provision of a conceptual bridge between familiar existing material and material that is new or technical appears to have some positive effect on the learning of a programming language. This might be due to the effects claimed by proponents of advance organizers (Ausubel, 1978; Mayer, 1979a, 1979b, 1981), or to the effects claimed by those supporting the provision of conceptual windows into the workings of computer languages (Sime and Fitter, 1978; Du Boulay and O'Shea, 1978, 1981; Howe and Ross, 1981). A more likely possiblity is that both strategies exercise some influence, singularly and in combination, towards an increase in learning. Further research is required in order to individually study these factors so as to be able to come to more meaningful conclusions.

In terms of the present study, the performance of the advance organizer group on the delayed posttest was both unexpected and puzzling. It is possible that the advance organizer model provided an initial learning advantage which decreased over the period between the two posttests. The performance of both the post-organizer and the no-model groups remained fairly steady over this same period (see Table 2). One interpretation of these results favourable to the present study, is that the provision of a graphic model as an advance organizer does have a significant effect on initial performance compared to a post-organizer model or to no model at all. If one wishes to pursue this interpretation, the problem of learning permanence becomes one of including review and practice over the long term within the instructional strategy. The author is not aware of any research addressing the effects of time and forgetting on the learning from graphic and concrete models, and certainly this is one area in which further research is called for.

Another interpretation of the unexpected results is that the application of graphic models does not have a more positive effect over a realistic and practical time interval, irrespective of any continuing review or practice strategies. In other words, though it might have been advantageous to provide the advance organizer group with review and practice, it might have been just as advantageous to provide the other two groups with such support activities The question then is whether or not review and practice by all three groups would have led to more effective and more permanent learning by any one. This again is an area in which further research is called for.

The review of literature has enumerated inconsistent conclusions from research into both advance organizers and problem solving. Such inconsistency could stem from problems in methodology and, indeed, there has been much criticism of the manner in which studies are both designed and carried out (Krasnor and Mitterer, undated; Sheil, 1981).

The present study suffered from several design and operational flaws. The performance measures were designed by the author, and pilot tested prior to their use in the study. However, because the measures were intended to evaluate two categories of learning (Comprehension/ Retention and Transfer), a more intensive pilot test should have been conducted in order that the independence of these categories could be ascertained through factor analysis. The fact that these categories were found to be correlated and contributing equalling to a single factor, did limit the ability of the study to properly address the hypotheses. In combining these categories into one performance measure, it was not possible to draw inferential conclusions about the type of learning facilitated by the treatments. Inspection of group means does suggest that there may be a differential effect favoring Transfer over Comprehension/ Retention for Group A, and this is supported by prior research (Mayer, 1975, 1976, 1981). However, no conclusions can be drawn from this as no statistical support for this speculation was found.

Another problem stems from the definitions of both Comprehension/Retention and Transfer. The author defined these terms in a manner consistent with the definitions found in current research. Unfortunately, many of these definitions are less than precise, defining comprehension as the ability to answer questions on material directly covered in instruction, and transfer of learning as the ability to apply presented material to new problem solving situations (Mayer, 1981). A more exacting suite of definitions would have led to a more discriminating set of performance measures, and perhaps to the independence of the intended learning categories.

Operationally, the author would have liked to run subjects on an individual basis through their respective treatments, but limitations on time and equipment made this impossible. The fact that subjects worked through the material in pairs introduced a possibly new variable and limits the generalizability of the results. In addition, the author would have preferred larger cell sizes to help offset the effects of the great range of individual variablity found in programming research (Sheil, 1981). The relatively small number of subjects involved certainly did not help in the discovery of any differences caused by treatment alone. However, the significant differences found on the immediate posttest between the advance organizer group and the two other groups, in spite of the small cell sizes, could be interpreted as partially supportative of advance organizer and conceptual model theory.

Conclusion

This study intended to ascertain the effectiveness of an instructional strategy for teaching abstract programming concepts to novice programmers. In so doing, it concluded that the use of graphic models of these abstract concepts does have a significant effect on immediate performance. In the longer run, however, this effect was found to be negligable. Such a drop in performance could be due to rapid forgetting, and it was suggested that because little research in this area has addressed the problem of forgetting, perhaps further study could be undertaken into the effect of review and practice on learning from graphic models.

Research has established, to a point, the efficacy of using concrete and graphic models within instructional strategies directed towards novice programmers. As the author previously noted, virtually all of this research has utilized either concrete or two-dimensional paper models. This study was an attempt to pursue this line of investigation into a CAL environment. Because instruction within CAL environments is becoming more and more an established reality, the author believes that the use of computer-presented graphic models should be pursued as a possibly fruitful area of research.

By combining the interactive, graphic capabilities of the computer with the significant gains in performance from conceptual models reported in the literature, the author suggests that a powerful and effective learning resource could be developed. But before the development of such a resource, much research is required into the effect of conceptual models (presented concretely or graphically) on different categories of learning, and over different periods of time. Only in this manner, can the efficacy of this instructional strategy be established.

REFERENCES

Abelson, H. (1982). LOGO programming for the Apple II. Peterborough, N.H., Byte-McGraw Hill.

Abelson, H. and Di Sessa, A. (1976). Student science training programs in mathematics, physics, and computer science. Final report to the N.S.F., Cambridge, Mass., A.I. Laboratory, M.I.T.

Alesandrini, K. L. and Rigney, J. W. (1981). Pictorial presentation and review strategies in science learning. Journal of Research in Science Teaching, 18, 5, 463-472.

Anderson, J. R. (1980). Cognitive psychology and its implications. San Francisco, W. R. Freeman.

Anderson, J. R. (1981). Acquisition of cognitive skills. Technical Report, Department of Pyschology, Carnegie-Mellon University.

Austin, H. (1976). Teaching teachers LOGO. The Lesley experiments. A.I Memo No. 336, Cambridge, Mass., A.I. Laboratory, M.I.T.

Ausubel, D. P. (1960). The use of advance organizers in the learning and retention of meaningful verbal material. Journal of Educational Psychology, 51, 267-272.

Ausubel, D.P. (1963). The psychology of meaningful verbal learning. New York, Grune and Stratton.

Ausubel, D. P. (1968). Educational psychology: A cognitive view. New York, Holt, Rinehart, and Winston.

Ausubel, D. P. (1978). In defence of advance organizers: A reply to the critics. Review of Educational Research, 48, 2, 251-257.

Ausubel, D.P. and Fitzgerald, D. (1961). The role of discriminability in meaningful verbal learning and retention. Journal of Educational Psychology, 52, 266-271.

Ausubel, D.P. and Youssef, M. (1963). The role of discriminability in meaningful verbal learning. Journal of Educational Psychology, 54, 331-336.

Badre, A. and Shneiderman, B. (1982). Directions in human/computer interaction. Norwood, N. J., Ablex.

Bamberger, J. (1972). Developing a musical ear: A new experiment. LOGO Memo No. 6, Cambridge, Mass., M.I.T.

Barnes, B. R. (1973). The effect of the position of organizers on the learning of structural anthropology materials in the sixth grade. Doctoral dissertation. University of Georgia, 1972. Dissertation Abstracts International, 33, 5027A-5028A.

Barnes, B. R. and Clawson, E. W. (1975). Do advance organizers facilitate learning? Review of Educational Research, 45, 637-659.

Barron, R. R. (1971). The effects of advance organizers upon the reception learning and retention of general science content. Final Report. Department of Health, Education and Welfare, Project No. 1B-030. ED 061 554.

Barron, R. R. (1980). A systematic research procedure, organizers and overviews: A historical perspective. Paper presented at the annual meeting of the National Reading Conference, San Diego.

Bartram, D. J. (1980). Comprehending spatial information: The relative efficiency of different methods of presenting information about bus routes. Journal of Applied Psychology, 65, 103-110.

Bernard, R. M., Petersen, C. H., and Ally, M. (1981). Can images provide contextual support for prose? ECTJ, 29, 2, 101-108.

Birch, A. (1980). LOGO curriculum for the Lincoln-Sudbury Schools. Mass., Unpublished manuscript.

Bloom, B. et al. (1956). Taxonomy of Educational Objectives. New York, David McKay and Company.

Bork, A. (1977). Learning through graphics. In R. J. Seidel and M. L. Rubin (Eds), Computers and Communication: Implications for Education. New York, Academic.

Bork, A. (1980). Textual taxonomy. Irvine, Cal., University of California, Educational Technology Center, Unpublished manuscript.

Bork, A. (1981). Learning with computers. Bedford, Mass., Digital Press.

Bork, A. Franklin, S. D., Von Blum, R., Katz, M., and Kurtz, B. L. (undated). Graphics and screen design for interactive learning. Unpublished manuscript.

Bransford, J. D. (1979). Human cognition. Belmont, Cal., Wadsworth Publishing.

Brooks, R. (1975). A model of human cognitive behaviour in writing code for computer programs. Doctoral dissertation, Carnegie-Mellon University.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies, 18, 543-554.

Brown, J. S. (1982). Narrative. In Seidel, R. J. et al, Computer Literacy.

Brown, J. S. and Rubinstein, R. (1974). Recursive functional programming for the student in the humanities and social sciences. Technical report no. 27a. Department of Information and Computer Science, University of California.

Cannara, A. B. (1976). Experiments in teaching children computer programming. Technical Report No. 271. Institute for Mathematical Studies in the Social Sciences, Stanford University.

Card, S. K., Moran, T. P., and Newell, A. (1983). The psychology of human-computer interaction. Hillsdale, N. J., Erlbaum.

Chait, S. (1978). An analysis of children's problem solving in a graphics oriented computer programming environment. MA Thesis, Montreal, McGill University.

Clawson, E. W. (1973). A comparison of the effects of organizers on the learning of structured anthropology materials in the third grade. Doctoral dissertation. University of Georgia. Dissertation Abstracts International, 33, 4788A.

Coombs, M. J. and Alty, J. L., (Eds). (1981). Computing skills and the user interface. London, Academic.

Coombs, M. J., Gibson, R. and Alty, J. L. (1982). Learning a first computer language: Strategies for making sense. International Journal of Man-Machine Studies, 16, 449-486.

Craik, K. (1943). The nature of explanation. Cambridge, Cambridge University Press.

Dooling, D. J. and Lachman, R. (1971). Effects of comprehension on the retention of prose. Journal of Experimental Psychology, 88, 216-222.

Dooling, D. J. and Mullet, R. L. (1973). Locus of thematic effects on retention of prose. Journal of Experimental Psychology, 97, 404-406.

Du Boulay, B. (1978). Learning primary mathematics through programming. Ph. D Thesis, Department of Artificial Intelligence, Edinburgh, University of Edinburgh.

Du Boulay, B. and O'Shea, T. (1978). Seeing the works: A strategy for teaching interactive programming. D. A I. Working Paper No. 28, Department of Artificial Intelligence, Edinburgh, University of Edinburgh.

Du Boulay, B. and O'Shea, T. (1981). Teaching novices programming. In M. J. Coombs and J. L. Alty (Eds), Computing Skills and the User Interface. London, Academic.

Du Boulay, B and Howe, J. A. M. (1981). Re-learning mathematics through LOGO: Helping student teachers who don't understand mathematics. In J. A. M. Howe and Ross, P. M., (Eds) Microcomputers in Secondary Education. London, Kogan Page.

Du Boulay, B., O'Shea, T, and Monk, J. (1981). The black box inside the glass box: Presenting computing concepts to novices. International Journal of Man-Machine Studies, 14, 237-249.

Duchastel, P. C. (1978). Illustrating instructional texts. Educational Technology, November.

Dywer, F. M. (1978). Strategies for improving visual learning. State College, P. A., Learning Services.

Feurzeig, W., Papert, S., Bloom, M., Grant, R., and Solomon, C. (1969). Programming language as a conceptual framework for teaching mathematics. Technical Report No. 1889, Bolt, Beranek, and Newman, November.

Feurzeig, W. and Lukas, G. (1975). The use of dribble files as instructional aids. LOGO Working Paper, Bolt, Beranek, and Newman.

Gagne, R. M. (1977). The conditions of learning. 3rd ed. New York, Holt, Rinehart, and Winston.

Glennan, T. K. (1967). Issues in the choice of development policies. In Marschak, T. et al, Strategies for R and D: Studies in the Microeconomics of Development. New York, Springer-Verlag.

Gnanadesikan, R. (1977). Methods for statistical data analysis of multivariate observations. New York, Wiley.

Goldenberg, P. (1980). Special technology for special children. Cambridge, Mass., M. I. T. Press.

Green, T. R. G. (1980). Programming as a cognitive activity. In H. T. Smith and T. R. G. Green, (Eds) Human Interaction with Computers. London, Academic.

Greeno, J. G. (1973). The structure of memory and the process of problem solving. In R. Solsa, (Ed) Contemporary Issues in Cognitive Psychology. Washington, Winston.

Greeno, J. G. (1980). Trends in the theory of knowledge for problem solving. In D. T. Tuma and F. Reif, (Eds) Problem Solving and Education: Issues in Teaching and Research. Hillsdale, N. J., Erlbaum.

Hartley, J. and Davies, I. K. (1976). Preinstructional strategies: The role of pretests, behavioural objectives, overviews, and advance organizers. Review of Educational Research, 46, 239-265.

Hilgard, D. and Bower, G. (1966) Theories of learning. New York, Meredith.

Hillis, D. (1975). Slot machine: Hardware manual. LOGO Working Paper No. 39. Cambridge, Mass., M. I. T. Press.

Hoc, J-M. (1977). Role of mental representation in learning a programming language. International Journal of Man-Machine Studies, 9, 87-105.

Holliday, W. G. (1976a). Teaching verbal chains using flow diagrams and texts. A. V. Communication Review, 24, 63-78.

Holliday, W. G. (1976b). Conceptualizing and evaluating learner aptitudes related to instructional stimulii in science education. Journal of Research in Science Teaching, 13, 101-111.

Howe, J. A. M. (1979). Learning through model building. In D. Michie, (Ed) Expert Systems in the Micro-electronic Age. Edinburgh, Edinburgh University Press.

Howe, J. A. M. and Ross, P. (1981). Moving LOGO into a mathematics classroom. In J. A. M. Howe and P. Ross, (Eds) Microcomputers in Secondary Education. London, Kogan Page.

Jagodzinski, A. P. (1983). A theoretical basis for the representation of on-line computer systems to naive users. International Journal of Man-machine Studies, 18, 215-252.

Kelman, P. et al. (1983). Computers in teaching mathematics. Addison-Wesley.

Keppel, G. (1973). Design and analysis. New Jersey, Englewood Cliffs, Prentice-Hall, Inc.

Kernighan, B. W. and Plauger, P. J. (1974). Programming style. SIGCSE Bulletin, 6, 90-96.

King, W. A. (1975). A comparison of three combinations of text and graphics for concept learning. Navy Personnel Research and Development Center Technical Report 76-16. ED 112 936.

Krasnor, L. R. and Mitterer, J. O. (undated). LOGO and the development of general problem solving skills. Brock University. Unpublished manuscript.

Lawton, J. T. and Wanska, S. K. (1977). Advance organizers as a teaching strategy. Review of Educational Research, 47, 233-244.

Leron, U. (undated). Some problems in children's LOGO learning. Israel, University of Haifa. Unpublished manuscript.

Levie, W. H. and Lentz, R. (1982). Effects of text illustrations: A review of research. ECTJ, 30, 4.

Levin, J. R. (1981). On functions of pictures in prose. In F. J. Pirozzolo and M. C. Wittrock, (Eds) Neurophysiological and Cognitive Processes in Reading. New York, Academic.

Loewe, H. (1979). Einfurhrung in die Lernpsychologie des Erwachsenalters. In W. Dehning, H. Essig, and S. Maass, The Adaptation of Virtual Man-computer Interfaces to User Requirements in Dialogs. Berlin, Springer-Verlag.

MacDonald-Ross, M. (1977). Graphics is texts. Review of Research in Education, 5, 49-85.

MacDonald-Ross, M. (1978). Language in texts. Review of Reseach in Education, 6, 229-275.

Mayer, R. E. Different problem solving competencies established in learning computer programming with and without meaningful models. Journal of Educational Psychology, 1975, 67, 725-734.

Mayer, R. E. (1976). Some conditions of meaningful learning for computer programming: Advance organizers and subject control of frame order. Journal of Educational Psychology, 68, 143-150.

Mayer, R. E. (1977). Different rule systems for counting behaviour acquired in meaningful and rote contexts of learning. Journal of Educational Psychology, 69, 537-546.

Mayer, R. E. (1979a). Can advance organizers influence meaningful learning? Review of Educational Research, 49, 371-383.

Mayer, R. E. (1979b). Twenty years of research on advance organizers: Assimilation theory is still the best predictor of results. Instructional Science, 8, 133-167.

Mayer, R. E. (1979c). A psychology of learning. Basic Communications of the A. C. M., 22, 11, 589-593.

Mayer, R. E. and Bromage, B. (1980). Different recall protocals for technical text due to advance organizers. Journal of Educational Psychology, 72, 209-225.
Mayer, R. E. (1981). The psychology of how novices learn computer programming. A. C. M. Computer Surveys, 13, 1, 121-141.

Mayer, R. E. (1982). Contributions of cognitive science and related research in learning to the design of computer literacy curricula. In Seidel, R. J. et al, Computer Literacy.

McKenzie, J., Elton, R. B., and Lewis, R., (Eds). (1978). Interactive computer graphics in science teaching. Chichester, Ellis Harwood.

Miller, L. A. (1974). Programming by non-programmers. International Journal of Man-Machine Studies, 6, 237-260.

Miller, L. A. (1975). Naive programmer problems with specification of transfer-of-control. Proceedings of the A F I P S National Computer Conference, 44, 657-663.

Mills, H. D. (1975). How to write correct programs and know it. Proceedings of the International Conference on Reliable Software, IEEE and Association for Computing Machinery, Los Angeles. ACM, New York.

Moore, M. V. and Nawrocki, L. H. (1978). The educational effectiveness of graphics displays for computer-assisted-instruction. Arlington, VA., Army Research Institute for the Behavioural and Social Sciences. ED 169 917.

Moore, M. V., Nawrocki, L. H., and Simutis, Z. M. (1979). The instructional effectiveness of three levels of graphic displays for computer-assisted-instruction. Arlington, VA., Army Research Institute for the Behavioural and Social Sciences, Technical Paper No. 359. ED 178 057.

Moore, D. W. and Readence, J. E. (1979). A meta-analysis of graphic organizers on learning from text. Paper presented at the annual meeting of the National Reading Conference, San Antonio.

Moran, T. P. (1981). The Command Language Grammer: A representation for the user interface of interactive computer systems. International Journal of Man-Machine Studies, 14, 3-50.

Myers, W. (1980). Computer graphics: The human interface. Computer, 13, 6, 45-54.

Newell, A. and Simon, H. A. (1972). Human problem solving. Englewood Cliffs, N. J., Prentice-Hall.

Nie, N. H., Hull, C. H., Jenkins, J. G., Steinbrenner, K., and Brent, D. H. (1975). SPSS: Statistical package for the social sciences. New York, McGraw-Hill.

Papert, S. (1971a). Teaching children to be mathematicians vs teaching about mathematics. LOGO Memo No. 4.

Papert, S. (1971b). Teaching children thinking. LOGO Memo No. 2. Cambridge, Mass., M. I. T. Press.

Papert, S. (1973). Uses of technology to enhance education. LOGO Memo No. 8. Cambridge, Mass., M. I. T. Press.

Papert, S. (1976). Some poetic and social criteria for education design. LOGO Memo No. 17. Cambridge, Mass., M. I. T. Press.

Papert, S. (1978). Computer-based microworlds as incubators for powerful ideas. Cambridge, Mass., M. I. T. Press.

Papert, S. and Solomon, C. (1972). Twenty things to do with a computer. Educational Technology, 12, 4, 9-18.

Pask, G. (1976). Styles and strategies of learning. British Journal of Educational Psychology, 46, 128-148.

Polya, G. (1971). How to solve it. Princton University Press.

Polya, G. (1965). Mathematical discovery; Volumns I and II. New York, John Wiley.

Resnick, L. B. and Ford, S. (1980). The psychology of mathematics learning. Hillsdale, N. J., Erlbaum.

Rigney, J. W. and Lutz, K. A. (1976). Effect of graphic analogies of concepts in chemistry on learning and attitude. Journal of Educational Psychology, 68, 305-311.

Royer, J. M. and Cable, G. W. (1975). Facilitated learning in connected discourse. Journal of Educational Psychology, 67, 116-123.

Royer, J. M. and Cable, G. W. (1976). Illustrations, analogies, and facilitated transfer in prose learning. Journal of Educational Psychology, 68, 205-209.

Scandura, J. M. and Wells, J. N. (1967). Advance organizers in learning abstract mathematics. Am. Educational Research Journal, 4, 295-301.

Schulz, R. W. (1967). The role of cognitive organizers in the facilitation of concept learning in elementary school science. Doctoral dissertation, Purdue University, 1966. Dissertation Abstracts, 27, 3784 A.

Schutz, R. E. The nature of educational development. Journal of Research Develop. Educ., 3, 2, 39-64.

Schwartz, N. H. and Kulhavy, R. W. (1981). Map features and the recall of discourse. Contemporary Educational Psychology, 6, 151-158.

Seidel, R. J., Anderson, R. E., and Hunter, B. (1982). Computer literacy: Issues and directions for 1985. New York, Academic Press.

Shneiderman, B. (1976). Exploratory experiments in programmer behaviour. International Journal of Computer and Information Sciences, 5, 2, 123-143.

Shneiderman, B. (1980). Software psychology, human factors in computer and information systems. Cambridge, Mass., Winthrop.

Shneiderman, B., Mayer, R. E., McKay, D., and Heller, P. (1977). Experimental investigations of the utility of detailed flowcharts in programming. Communications of the A. C. M., 20, 373-381.

Shneiderman, B. and Mayer, R. E. (1979). Syntactic/semantic interaction in programmer behaviour: A model and experimental results. International Journal of Computer and Information Sciences, 8, 3, 219-238.

Sime, M. E., Arblaster, A., and Green, T. R. G. (1977). Structuring the programmer's task. Journal of Occupational Psychology, 50, 205-216.

Sime, M. E. and Fitter, M. J. (1978). Computer models and decision making. In P. B. Warr, (Ed) Psychology at Work (2nd ed). London, Penquin.

Simon, H. A. (1969). The sciences of the artificial. Cambridge, Mass., M. I. T. Press.

Simon, H. A. (1980). Problem solving and education. In D. T. Tuma and F. Reif, (Eds) Problem Solving and Education: Issues in Teaching and Research. Hillsdale, N. J., Erlbaum.

Sless, D. (1981). Learning and visual communication. London, Croom Helm.

Solomon, C. (1976). Problem solving in an anthropomorphic computer. Master's Thesis. Boston University.

Soloway, E., Lochhead, J., and Clement, J. (1982). Does computer programming enhance problem solving ability? Some positive evidence on algebra word problems. In Seidel, R. J. et al, Computer Literacy.

Soloway, E. et al. (1982). What do novices know about programming? In A. Badre and B. Shneiderman, (Eds) Directions in human/computer interaction. Norwood, N. J., Ablex.

Statz, J., Folk, M., and Seidman, R. (1973). Final report. Syracuse University LOGO project. Syracuse University.

Watt, D. (1982). LOGO in the schools. BYTE, 7, 116-134.

Weaver, F. and Suydam, M. (1972). Meaningful instruction in mathematics education. Mathematics Education Reports.

Weinberg, G. M. (1971). The psychology of computer programming. New York, Van Nostrand.

Weisberg, J. S. (1970). The use of visual advance organizers for learning earth science concepts. Journal of Research in Science Teaching, 7, 161-165.

Weyer, S. A. and Cannara, A. B. (1975). Children learning computer programming: Experiments with languages, curricula, and programmable devices. Technical Report No. 250, Institute for Mathematical Studies in the Social Sciences, Stanford University.

Winn, W. O. (1980). The effect of block-word diagrams on the structuring of science concepts as a function of general ability. Journal of Research in Science Teaching, 17, 3, 201-211.

Winn, W. O. (1981). Effect of attribute highlighting and diagrammatic organization on identification and classification. Journal of Research in Science Teaching, 18, 1, 23-32.

Winn, W. O. (1982). The role of diagrammatic representation in learning sequences, identification, and classification as a function of verbal and spatial ability. Journal of Research in Science Teaching, 19, 1 79-89.

Winn, W. O. and Holliday, W. G. (1981). Learning from diagrams: Theoretical and instructional considerations. Paper presented at the annual convention of the Association for Educational Communication and Technology, Philadelphia.

Woodward, E. L. (1967). A comparative study of teaching strategies involving advance organizers and post organizers and discovery and non-discovery techniques where the instruction is mediated by computer. Doctoral dissertation, Florida State University, 1966. Dissertation Abstracts, 27, 3787A.

Young, E. A. (1974). Human errors in programming. International Journal of Man-machine Studies, 6, 361-376.

Young, R. M. (1981). The machine inside the machine: Users' models of pocket calculators. International Journal of Man-machine studies, 15, 51-85.

Appendix A

Name: ______________________
(Please Print)

QUIZ

This quiz is designed to assess your understanding of the program on Logo word and list operations. Please try to answer each question as accurately as possible. There is no time limit.

1. Which Logo operations will shorten a list by one word?

____________________ and ____________________

2. Which Logo operation will add a word to the end of a list?

____

3. Which Logo operation will form a single list from several smaller lists?

____

4 Which Logo operations will output one of the words from a list of words?

____________________ and ____________________

5. Which Logo operation will add a word to the beginning of a list?

____

CONT'D/

For each of the following questions, you are given a word or list as input to a Logo operation. Indicate on the OUTPUT LINE, the NEW word or list output from this operation. (Use brackets and quotation marks where required.)

6. Input: ((Birds fly) (Rabbits run) (Fish swim))
   Operation: FIRST
   Output: ______________________________

7. Input: (2X 3Y 4Z)
   Operation: BUTFIRST
   Output: ______________________________

8. Input: ((Mary Jane) (Edward) (Billy Joe))
   Operation: LAST
   Output: ______________________________

9. Input: "Pears (Apples Plums Grapes)
   Operation: FPUT
   Output: ______________________________

10. Input: ((Sky blue) (Wine red) (Canary yellow))
    Operation: BUTLAST
    Output: ______________________________

11. Input: "WX "Y "Z
    Operation: WORD
    Output: ______________________________

12. Input: "Hearts (Spades Diamonds Clubs)
    Operation: LPUT
    Output: ______________________________

13. Input: (Rye) "Barley (Wheat Oats)
    Operation: SENTENCE
    Output: ______________________________

14. Input: "Sailing
    Operation: BUTFIRST
    Output: ______________________________

For each of the following questions, you are given a word or list as input and a NEW word or list as output. Indicate on the OPERATION LINE, the Logo OPERATION that output the new word or list.

15. Input: "ers (Crack)
    Output: (Crackers)
    Operation: ____________

16. Input: ((Silver spruce) (Red fir) (Weeping willow))
    Output: (Silver spruce)
    Operation: ____________

CONT'D/

17. Input: (Come out) (with) (your hands up.)
    Output: (Come out with your hands up.)
    Operation: ____________

18. Input: (3P 5Q 4R 2S)
    Output: (3P 5Q 4R)
    Operation: ____________

19. Input: "10 (40 30 20)
    Output: (40 30 20 10)
    Operation: ____________

20. Input: (Earth Air Fire Water)
    Output: (Air Fire Water)
    Operation: ____________

21. Input: "Mem "or "able
    Output: "Memorable
    Operation: ____________

22. Input: ((Green bean) (Red pepper) (Chick pea))
    Output: (Chick pea)
    Operation: ____________

23. Input "Shoot
    Output: "hoot
    Operation: ____________

Place a check mark beside the COMBINATION of Logo operations that will output the second word of a list. (Remember, Logo executes operations from RIGHT to LEFT.)

a) LAST BUTFIRST
b) FIRST BUTLAST
c) FIRST BUTFIRST
d) LAST BUTLAST

25. Place a check mark beside the COMBINATION of Logo operations that will output the first character of the last word of a list.

a) FIRST FIRST
b) LAST LAST
c) FIRST LAST
d) LAST FIRST

26. Place a check mark beside the COMBINATION of Logo operations that will output the third word from the end of a list.

a) FIRST BUTFIRST BUTLAST
b) FIRST BUTLAST BUTLAST
c) LAST BUTFIRST BUTFIRST
d) LAST BUTLAST BUTLAST

CONT'D/

For each of the following questions, you are given a word or list as input to a COMBINATION of Logo operations. Indicate on the OUTPUT LINE, the NEW word or list output from this combination of operations.

27. Input: (This is not hard)
Operations: FIRST BUTFIRST
Output: ____________________

28. Input: "Green "Apples
Operations: FIRST WORD
Output: ____________________

29. Input: ((Muffin) (Apple pie) (Chocolate cake))
Operations: LAST BUTLAST
Output: ____________________

30. Input: (The wine was sour.)
Operations: LAST BUTLAST BUTLAST
Output: ____________________

31. Input: ((2P 4Q) (3R) (2S 5T))
Operations: FIRST BUTFIRST BUTFIRST
Output: ____________________

32. Input: "Hours (Minutes Seconds)
Operations: LAST BUTFIRST LPUT
Output: ____________________

33. Input: (The) (Wizard) (of Oz)
Operations: BUTFIRST BUTLAST BUTLAST SENTENCE
Output: ____________________

34. Input: "Under (In On)
Operations: FIRST BUTFIRST FPUT
Output: ____________________

35. Input: (Stormy Weather)
Operations: FIRST FIRST
Output: ____________________

36. Input: "Ab (bra cada bra)
Operations: LAST LAST FPUT
Output: ____________________

37. Input: ((A) (B) (C))
Operations: BUTFIRST BUTFIRST BUTFIRST
Output: ____________________

38. Input: "na "tion "al
Operations: BUTLAST BUTLAST WORD
Output: ____________________

CONT'D/

For each of the following questions, you are given a word or list as input and a NEW word or list as output. Indicate on the OPERATIONS LINE, the COMBINATION of Logo operations that output the new word or list. Write the operations in the ORDER that would allow Logo to execute them.

39. Input: (Phantom of the Opera)
Output: the
Operations: ____________________

40. Input: (This is not so easy.)
Output: (not easy.)
Operations: ____________________

41. Input: "Blue "Monday
Output: B
Operations: ____________________

42. Input: "Eyes (Ears Nose Throat)
Output: (Eyes Ears Nose)
Operations: ____________________

43. Input: (A B) (C) (D E) (F)
Output: (C D E)
Operations: ____________________

44. Input: "Robbery (Great Train)
Output: (Train Robbery)
Operations: ____________________

45. Input: (Black cat Blue bird)
Output: (Black bird)
Operations: ____________________

46. Input: "Egg "shell
Output: s
Operations: ____________________

47. Input: (2X 3Y 6P 4R 3Q)
Output: (3Y 6P 3Q)
Operations: ____________________

48. Input: "C (N Z)
Output: N
Operations: ____________________

49. Input: ((Gold watch) (Silver dollar) (Copper penny))
Output: dollar
Operations: ____________________

50. Input: "Cape "so
Output: "apes
Operations: ____________________

/END

NAME: ____________________
(Please print)

QUIZ

This quiz is a shorter, altered version of the quiz you completed after viewing the program on Logo word and list operations. We would like you to complete it in order to determine how much you have retained over a period of several days. Thank you for your co - operation.

For each of the following questions, you are given a word or list as input to a Logo OPERATION. Indicate on the OUTPUT LINE, the NEW word or list output from this operation. (Use brackets and quotation marks where required.)

1. Input: ([Kites fly) (Ships sail) (Trains roll))
   Operation: FIRST
   Output: ______________________________

2. Input: ((Do Re) (Mi Fa) (So))
   Operation: BUTLAST
   Output: ______________________________

3. Input: "2P "5Q "3R
   Operation: WORD
   Output: ______________________________

4. Input: "Blue (Red Green Yellow)
   Operation: LPUT
   Output: ______________________________

5. Input: (Apples) "Oranges (Pears Plums)
   Operation: SENTENCE
   Output: ______________________________

For each of the following questions, you are given a word or list as input and a NEW word or list as output. Indicate on the OPERATION LINE, the Logo OPERATION that output the new word or list.

6. Input: ((Red oak) (White birch) (Silver spruce))
   Output: (Red oak)
   Operation: ____________

7. Input: (Wait) (for the) (bus here.)
   Output: (Wait for the bus here.)
   Operation: ____________

8. Input: "One (Four Three Two)
   Output: (Four Three Two One)
   Operation: ____________

9. Input: "2X "3Y "4Z
   Output: "2X3Y4Z
   Operation: ____________

10. Input "Show
   Output: how
   Operation: ____________

CONT'D/

11. Place a check mark beside the COMBINATION of Logo operations that will output the last character of the last word of a list. (Remember, Logo executes operations from RIGHT to LEFT.)

a) FIRST FIRST
b) LAST LAST
c) FIRST LAST
d) LAST FIRST

12. Place a check mark beside the COMBINATION of Logo operations that will output the second word from the end of a list.

a) LAST BUTFIRST
b) LAST BUTLAST
c) FIRST BUTLAST
d) LAST FIRST

13. Place a check mark beside the COMBINATION of Logo operations that will output the third word from the beginning of a list.

a) FIRST BUTFIRST BUTLAST
b) FIRST BUTFIRST
c) FIRST FIRST BUTFIRST
d) FIRST BUTFIRST BUTFIRST

For each of the following questions, you are given a word or list as input to a COMBINATION of Logo operations. Indicate on the OUTPUT LINE, the NEW word or list output from this combination of Logo operations.

14. Input: "Red "Grapes
Operations: FIRST WORD
Output: ______________________________

15. Input: "Ginger (Vanilla Chocolate)
Operations: LAST BUTFIRST LPUT
Output: ______________________________

16. Input: (The) (Duke) (of Kent)
Operations: BUTFIRST BUTLAST BUTLAST SENTENCE
Output: ______________________________

17. Input: "Down (Over Under)
Operations: FIRST BUTFIRST FPUT
Output: ______________________________

18. Input: "in "tern "al
Operations: BUTLAST BUTLAST WORD
Output: ______________________________

CONT'D/

For each of the following questions, you are given a word or a list as input and a NEW word or list as output. Indicate on the OPERATIONS LINE, the COMBINATION of Logo operations which output the new word or list.

19. Input: (From Here to Eternity)
Output: to
Operations: ______________________

20. Input: (This is not that clear.)
Output: (not clear.)
Operations: ______________________

21. Input: "Wave (The Third)
Output: (Third Wave)
Operations: ______________________

22. Input: (Red ball Blue beard)
Output: (Red beard)
Operations: ______________________

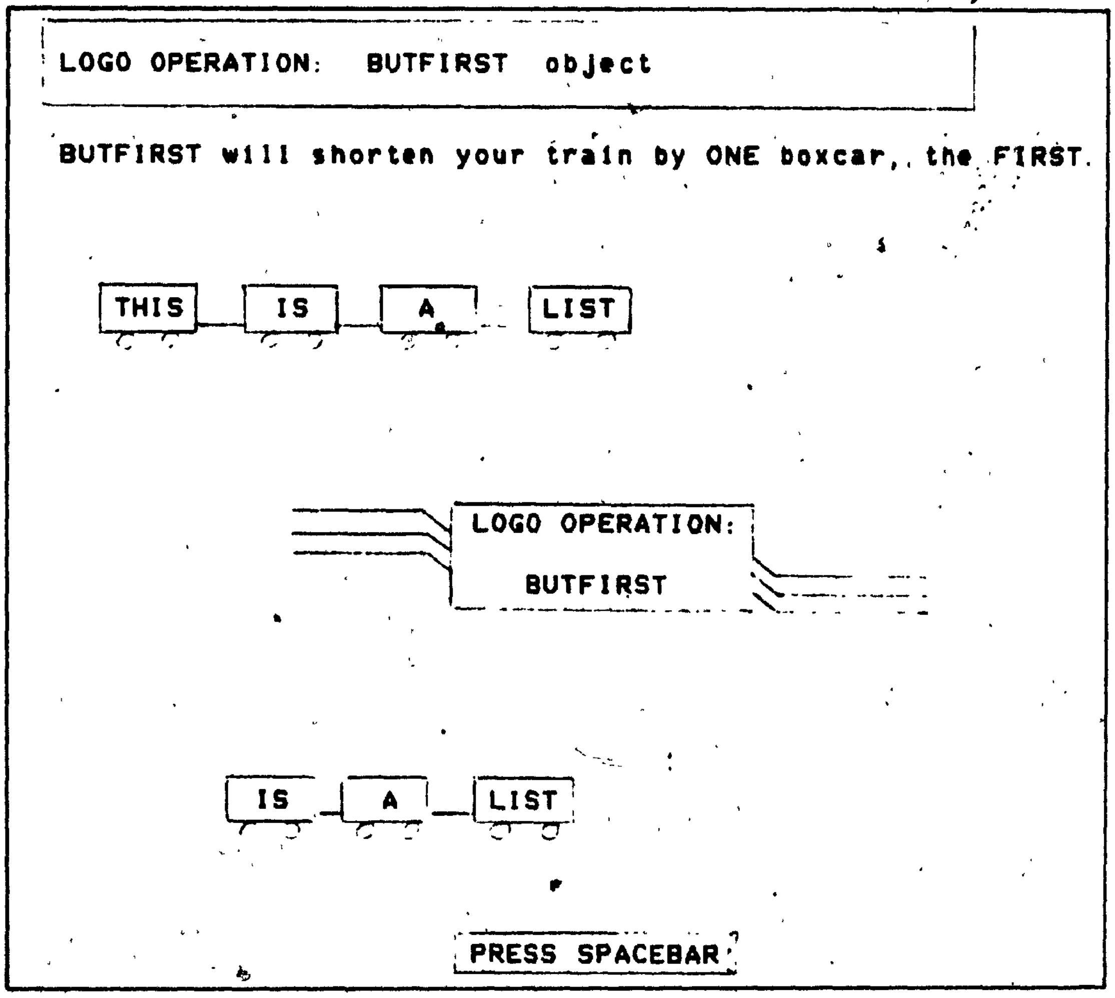
23. Input: "Paper "Dolls
Output: D
Operations: ______________________

24. Input: (2A 3B 6M 4Q 3S)
Output: (3B 6M 3S)
Operations: ______________________

25. Input: "Town "err
Output: "owner
Operations: ______________________

/END.

Appendix B



LOGO OPERATION: BUTFIRST object

BUTFIRST will shorten your train by ONE boxcar, the FIRST.