THE EFFECT OF COMPUTER PROGRAMMING EXPERIENCE ON MATHEMATICAL PROBLEM SOLVING ABILITY

by

Leah Paulette McCoy

Dissertation submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

DOCTOR OF EDUCATION

in

Curriculum and Instruction

APPROVED:

Norman R. Dodl, Chairman

John K. Burton

Larry J. Weber

Catherine A. Brown

James W. Garrison

Lee M. Wolfle

May, 1987

Blacksburg, Virginia

THE EFFECT OF COMPUTER PROGRAMMING EXPERIENCE ON MATHEMATICAL PROBLEM SOLVING ABILITY

by

Leah Paulette McCoy

Committee Chairman: Norman R. Dodl
Curriculum and Instruction

(ABSTRACT)

Five component problem-solving skills (general strategy, planning, logical thinking, algebraic variables, and debugging) were identified as common elements of both computer programming and mathematical problem solving. Based on the similarities of these general skills in specific contexts, a theory was generated that the skills would transfer and that experience in computer programming would cause an improvement in mathematical problem-solving achievement.

A path model was constructed to illustrate this hypothesized causal relationship between computer programming and mathematical problem-solving achievement. In order to control for other relevant variables, the model also included mathematics experience, access to a home computer, ability, socioeconomic status, and gender. The

model was tested with a sample of 800 high school students in seven southwest Virginia high schools.

Results indicated that ability had the largest causal effect on mathematical problem-solving achievement. Three variables had a moderate effect: computer programming experience, mathematics experience, and gender. The other two variables in the model (access to a home computer and socioeconomic status) were only very slightly related to mathematical problem-solving achievement.

The conclusion of the study was that there was evidence to support the theory of transfer of skills from computer programming experience to mathematical problem solving. Once ability and gender were controlled, computer programming experience and mathematics experience both had causal effects on mathematical problem-solving achievement. This suggests that to maximize mathematical problem-solving scores, a curriculum should include both mathematics and computer programming experiences.

ACKNOWLEDGMENTS

I wish to thank the many people who have assisted me during the planning and implementation of this dissertation.

My parents and my brother have supported and encouraged me, and put up with a lot of aggravation. My niece, , has been a source of inspiration.

, my Chairperson, has patiently shared his time and expertise. He has been a true advisor and friend.

My other committee members, , , , and , have also given valuable guidance and assistance.

My coworkers in the Microcomputer Lab have shared this entire process. They have been my best friends and worst critics throughout.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION AND RATIONALE

Computers are an integral part of today's world. Nearly every facet of life involves computers, from the supermarket check-out line to the Internal Revenue Service. Many jobs involve the use of computers in some capacity. Largely because of this widespread use (and the resulting demand from the public), schools are adopting the goal of "computer literacy" for their students.

Because computer education is still at a formative stage, "computer literacy" is not well defined. Some educators favor requiring computer literacy courses including programming for all students (Bitter & Camuse, 1984; Kulm, 1984; Luehrmann, 1981, 1984; Quinn, Kirkman, & Schultz, 1983). Others favor limiting computer literacy to machine operation and utilization of pre-programmed software (Bork, 1985; Hunter, Dearborn, & Snyder, 1983; Johnston, 1985; Laney, 1985; Lockheed & Mandinach, 1986). Still other educators discuss the problem at length and conclude only that we need more research-based information to make a decision about the content of computer literacy curricula (Battista & Steele, 1984; Kleiman, 1984; Lepper, 1985; Sheingold, Kane, & Endreweit, 1983). In order to use

a computer effectively in most software applications, no knowledge of programming is necessary. Only a small percentage of students will pursue careers as professional programmers. So, if we are to justify teaching computer programming to all secondary school students, some more generalizable benefit must be found.

One of the most prevalent claims is that computer programming improves problem-solving ability. This claim is supported by a small amount of empirical research (Foster, 1972; Hart, 1982; Johnson & Harding, 1979; Mayer, Dyck, & Vilberg, 1986; Nowaczyk, 1984; Wells, 1981) and also by a number of untested, but logically consistent arguments (Lepper, 1985; Nickerson, 1982; Papert, 1980; Shneiderman, 1985; Weinberg, 1971).

Since problem solving is a major focus of this study, the definition of a problem is important. The following are taken from the mathematics literature.

> "An individual is faced with a problem when he encounters a question he cannot answer or a situation he is unable to resolve using the knowledge immediately available to him. He must then think of a way to use the information at his disposal to arrive at the goal, the solution of the problem" (Kantowski, 1977, p. 163).
>
> "There are many interpretations of the meaning of the word problem. To some, particularly those who work with children at the early elementary level, problems are verbal or "word" problems. To others, they are nonroutine situations, such as puzzles, logic problems or those requiring a bit of insight or creative thinking often found in "challenge" sections of textbooks. To still

> others, they are applications, or real-world problems. A problem can be "any of the above." What is a problem for one student may not be a problem for another--or even for the same student at a later time. For most students, textbook word problems are problems at first but they become exercises once students learn an algorithm or method to solve them. What makes a problem a problem is that, at least at first, there is some uncertainty about how to find a solution. Students have at their disposal, or can easily find, all the necessary facts but they are unsure of how to put them together to complete the task at hand" (Kantowski, 1982, p. 75).
>
> "A problem is a situation in which an individual or group is called upon to perform a task for which there is no readily accessible algorithm which determines completely the method of solution" (Lester, 1980, p. 287).
>
> "A true problem requires a search for possible solutions because no solution is readily apparent" (Resnick & Ford, 1981, p. 148).

Polya (1957) who is one of the early writers on the topic defines problem solving as follows.

> "Solving a problem is finding the unknown means to a distinctly conceived end. If the end by its simple presence does not instantaneously suggest the means, if therefore, we have to search for the means, reflecting consciously how to attain the end, we have to solve a problem" .

For the purposes of this study "problem solving" is defined as a case in which: a problem is presented for which the problem solver wants a solution; a solution is not readily available; and the problem solver must use existing knowledge in new and different ways in constructing a solution to the problem.

Much of the logical justification for the theory that computer programming skills may transfer to other problem-solving activities is based on the similarity of the two exercises. The process of writing computer programs can be considered a problem-solving activity, because it satisfies the definition of problem solving (Kulm, 1984). Since courses in programming are necessarily activity oriented, they provide extensive practice in problem solving in a specific context. Therefore, it is possible that computer programming may improve general problem-solving ability.

Mathematical problem solving is an area of interest because a goal of mathematics education is to prepare students for the future (Gagne, 1983; Kilpatrick, 1969; Underhill, 1977). Since the future is difficult to foresee, students must be taught to apply mathematical skills and concepts to novel situations. The National Council of Teachers of Mathematics (1980) has targeted problem solving as the "focus" of school mathematics of the 1980's. Similarly, the National Council of Supervisors of Mathematics (1977, p. 2) stated that "learning to solve problems is the principal reason for studying mathematics." Nearly every mathematics curriculum K-12 includes some goals aimed at developing mathematical problem-solving skill. Even with this attention, students' mathematical problem-solving skills are often poor. In a review of the results of the second National Assessment of Educational

Progress, Carpenter, Corbitt, Kepner, Lindquist, and Reys (1980) noted the widespread failure of students on items dealing with solving mathematical problems.

In this chapter the relationship of mathematical problem solving and computer programming problem solving is examined. First, the human information processing system is described. Then a general problem-solving model, based on the theories of Polya (1957, 1962, 1965) and of Newell and Simon (1972), is presented. The process of problem solving in mathematics and in computer programming is examined in the context of this model. Next, the concept of transfer is discussed. Finally, potentially transferable skills which are critical for problem solving in both mathematics and computer programming are explored.

The Human Information Processing System

The information processing system model is a way of describing human cognition (Klatzky, 1980; Mayer, 1983; Zechmeister & Nyberg, 1982). According to this view, stimuli are received from the outside world via receptors, which are the senses. The information is held temporarily in the sensory registers. This is a very limited storage, both in space and in time. Processing begins at this point with some recognition analysis of the stimulus, involving

accessing long term memory in attempt to match the current stimulus with existing knowledge via a prototype.

The control processor is the human consciousness, or working area. It contains three main parts: a mechanism to access information in long term memory, the short term memory, and a mechanism to encode information into long term memory. It accesses long term memory for prototypes and schema to bring order to new information. The short term memory holds a limited amount of information for a limited time while it is being actively processed. The encoder functions to control the encoding and retrieval of information from long term memory.

The long term memory is a vast area where information is stored. This information is stored in a complex network at the time of encoding. It is accessed by the sensory registers and the control processor, and receives input from both of these components. Retrieval of information is dependent on how it was encoded, using semantic information and cues.

The response generator is a component accessed mainly by short term memory which generates an external response. The effectors are the means by which a response is sent to the outside world, such as speech.

Expert/Novice Research

Experts and novices in a specific skill area store specific information differently in long-term memory. This difference involves both quality and quantity of information, with the expert having a much more efficient storage system and thus a more efficient retrieval system. In fact, retrieval of specific information is usually automatic for the expert.

Early research in this area was involved in studying chess players (Chase & Simon, 1973; DeGroot, 1965). It was found that chess masters could easily reproduce entire chess boards from memory, while novices could only reproduce the positions of a few pieces. This is evidence of the chunking ability of the experts. Similar evidence has been reported for players of go (Reitman, 1976) and for electronic technicians (Egan & Schwartz, 1979).

Two studies have reported evidence of chunking in expert computer programmers. Adelson (1981) defined novices as students who had had one course, and experts as computer science teaching assistants. She presented lines of program code in normal and in random order. She found that novices used a syntax-based organization and experts used a more functional organization.

In a similar study, McKeithen et al. (1981) found that beginners organized programming concepts by common-language

associations. Intermediate programmers mixed programming and common-language associations, and experts' organization was based on programming knowledge. These studies provide evidence of a difference in organization of programming concepts and skills by experts and novices.

In slightly different terms, Linn (1985) describes the levels of expertise for students learning computer programming. She calls this model the "Chain of Cognitive Accomplishments from Learning Programming." The three levels are language feature, design skills, and problem-solving skills. She theorizes that students at different levels will perform differently both within the programming context and in using the programming context to develop general problem-solving skills.

A Problem Solving Model

Polya

A classic model of problem solving presented by Polya (1957), includes the following steps:

1. Understanding the Problem,
2. Devising a Plan,
3. Carrying Out the Plan, and
4. Looking Back. (See Figure 1.)

Even though the model appears linear, in practice it is recursive. Current and previous steps are often repeated

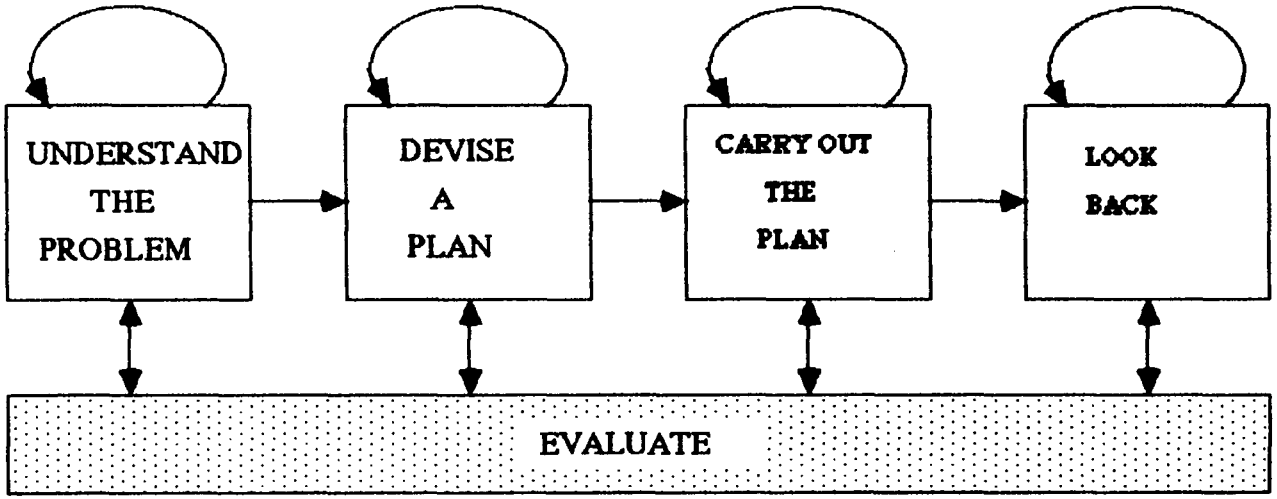

Figure 1. Polya's Problem Solving Model

as evaluation occurs throughout the process. This model can be seen as an overall strategy that drives the human information processing system during problem solving. It is stored in long term memory and is accessed when a problem is encountered. It functions as a general framework within which problem solving is carried out. As such, it is a general strategy for solving a problem--whether mathematics, or computer programming, or some other content area. The specifics within the four steps may vary slightly, but the overall model applies to all areas.

Newell and Simon

Another classic theory of human problem solving (Newell & Simon, 1972) can be used to explain the specific internal processes that occur during problem solving. This model characterizes problem solving as a search through the memory system for potential solutions. This search is defined by the external task environment called the problem and the internal representation called the problem space. The problem solver's long-term memory, which consists of a structured representation of experiences and operations, is the system searched. This structure is hierarchial and is believed to be in the form of conditional productions (Brooks, 1977; Mayer, in press; Resnick & Ford, 1981). Once a schema is returned from long term memory and evaluated in terms of the problem space in short term

memory, possible solutions and/or methods are ranked and the most favorable one is chosen. It is then evaluated in terms of the original problem. The process may be repeated recursively until a satisfactory solution is reached.

An Integrated Model

While these two models may appear different, they are actually quite compatible. The Polya model is a system for the control processor that organizes the problem conditions and sets up the search. The Newell and Simon theory explains the specifics of the internal information processing that occurs as a search is executed. Therefore, the two theories are complementary, with Polya providing the overall strategy, and Newell and Simon explaining the specific internal processes.

The Problem Solving Process

Understanding the Problem

At Step 1, the problem solver seeks to understand the problem. The meanings of the problem are internalized and construction of a problem space for that problem begins. Different problem solvers may construct very different problem spaces for the same problem. If an incorrect problem space is constructed, the problem solver will have

to return to this step at some point and repeat the process of understanding the problem.

Schemata which are used at this step usually contain linguistic and/or factual information. The linguistic knowledge is necessary in order to understand the language of the problem. Since most problems are in written form, this is equivalent to reading. Research has found that verbal skill significantly impacts problem solving (Foster, 1972; Sauter, 1986; Webb, 1979).

Necessary factual knowledge may vary widely according to the specific information in the problem. A broad general knowledge base has been found to facilitate problem solving (Gagne, 1982; Michener, 1978; Sheil, 1981). There is little difference at this step in solving a mathematics or a programming problem, except that the factual knowledge varies both within and between the two types of problems.

Devising a Plan

At Step 2, the problem solver devises a plan for solution. Polya (1957) describes planning as "provisional and plausible reasoning" (p. 113). Mayer (1981) also mentions the tentative nature of planning, defining it as construction of an abstract form of a potential solution.

Planning begins as a search of existing knowledge in an attempt to locate schema to identify the problem type and to suggest an initial solution procedure. In

mathematics, Resnick and Glaser (1976) identify strategies of problem detection as a critical aspect of problem solving. Gagne (1982) also mentions the need for schemata to help in translating "concretely stated" problems to mathematical form. Hinsley, Hayes, and Simon (1977) extensively studied algebra word problems and concluded that they could be classified into 18 distinct story schemata. During this stage of the solution, the problem solver searches long term memory for a schema that matches the conditions of the problem as closely as possible. That schema is then used as a basis for searching for a tentative plan for solution.

In computer programming, the programmer similarly searches for a schema that helps to plan the solution. These schemata contain conditions along with processes (Nutter & Hassell, 1985). They also may contain algorithms or chunks of code for particular purposes (Adelson, 1981; McKeithen, Reitman, Reuter, & Hirtle, 1981; Mayer, 1985).

In both mathematical problem solving and computer programming, after schemata are located and possible solution methods are examined, the problem solver integrates one or more methods into a design for solving the problem. Few problems are solved with only one operation so this design stage is extremely important in planning a solution. In both mathematics and computer programming, the design often consists of a method or a

list of the steps to be accomplished. Polya's description of a design is "knowing at least in outline, which calculations, computations, or constructions we have to perform in order to obtain the unknown" (1957, pp. 8-9).

In programming, where planning and design are strongly emphasized, it is common to plan the solution on a flow chart or a structure chart. Shneiderman (1976) says that program planning, using diagrams, flow charts, mathematics notation, and programming language notation, improves thinking. In programming, students are taught to structure their solution plans. This requires that the problem be decomposed, and the solution be planned in a structured design (Lepper, 1985; Nickerson, 1982). The concept of problem decomposition into smaller, and easier, subproblems is a potentially useful tool in any kind of problem solving.

At the planning stage, it may be necessary to return to the understanding stage several times to re-analyze the problem. Also, it is likely in both types of problems that no schema exactly fits the problem. In this case, the available related schemata are searched, and the best available plan is selected on a tentative basis. This is followed by evaluation, refinement, and repetition of the process until an ultimate acceptable solution is achieved (Newell & Simon, 1972).

Carrying Out the Plan

The third step is carrying out the plan. This step requires specific procedural knowledge. In mathematics, this often involves arithmetic operations or translating the plan to an algebraic equation and solving it. In either case, the procedural knowledge required is quantitative skill. The problem solver must know how to perform calculations, how to translate from words to equations, and how to solve equations.

In computer programming, the product is a program that is properly structured and produces a desired result. The procedural knowledge involved at this step in computer programming is translation of the design to code. Just as solution of numerical or algebraic expressions involves a search in long term memory for a rule or algorithm, the code generation process involves a search for a rule that fits the conditions. Brooks (1977) says that an expert computer programmer may have internal access to tens or hundreds of thousands of rules for code generation.

As the plan is carried out, the problem solver checks his execution, whether he calculated, wrote an equation and solved it, or wrote a program. Depending on the outcome, the problem solver may go back to step 2 and examine the plan, or all the way back to step 1 to check his understanding of the problem. In computer programming, the computer itself assists with the procedural checking, and

gives error messages for syntactic program errors. The problem solver then debugs the program by correcting the syntax error and repeating the step. It is clear from this model that checking is a necessary activity in problem solving. The "built-in" feedback mechanism of the computer is an additional aid in the checking process.

Looking Back

Step 4 is looking back. While evaluation is an integral part of each step of the process, this final step involves overall evaluation and reflection. In mathematics, the proposed solution is checked back in the original problem. If it fits, then this step involves building schemata in long term memory to be available if this problem or similar ones are encountered in the future. If the solution does not seem reasonable, checks are made at any or all previous steps. The entire process or parts of it are repeated until an acceptable solution is attained.

In computer programming, this works much the same way. A program may be syntactically correct and run without error, but may produce incorrect results. The problem solver then must check all previous steps and attempt to discover and correct his "logic error." This second type of debugging usually involves revision of the design and

execution of the plan and also may require a return to Step 1 to clarify the meaning of the problem.

Another part of looking back is reflecting on the solution to see if it could have been achieved in a more efficient or more elegant way (Kilpatrick, 1967; Nickerson, 1982). This is true in both mathematical problem solving and in computer programming, where the problem solver carefully examines the solution method and the final solution. By working backward, another solution method may be discovered. This alternate method is also stored in long term memory for future use.

In summary, the integrated problem-solving model, incorporating the theories of Polya and of Newell and Simon has been used as a framework for describing the problem-solving process. The four steps of understanding the problem, devising a solution, carrying out the plan, and looking back have been shown to be descriptive of both mathematical problem solving and computer programming.

Transfer

As stated earlier, the premise for expecting a positive effect on mathematical problem solving from computer programming is that certain skills and techniques will transfer from one task to the other. Transfer refers

to the application of knowledge and skills learned in one task to a different task.

Historically, transfer has been a topic of interest to educators and psychologists. The first 200 years of American education, beginning with the Latin school movement, were primarily based on the approach of faculty psychology (Hudgins, 1977). This is the idea that training in Latin, Greek, and mathematics would train the mind and develop such faculties as memory and reason (Hudgins, 1977; Mayer, in press). This theory gradually fell out of favor and was replaced by a totally opposite viewpoint, Thorndike's theory of identical elements (Thorndike & Woodworth, 1901). This position is that transfer is possible only if the two tasks contain identical elements. Based on a study of high school curricula, Thorndike (1924) concluded that knowledge and skills were the important result of study of any particular subject, and that no general benefit should be expected.

Osgood's (1949) surface is a classic transfer model which shows that maximum positive transfer occurs when the stimuli and the responses of the original and the new task are identical. Lesser degrees of transfer occur when the tasks are more dissimilar. Hudgins (1977) differentiates between near and remote transfer, depending on the similarity of the original and the new learning task. Several authors agree that near transfer is much more

likely (Frederiksen, 1984; Hudgins, 1977; Kilpatrick, 1978; Underhill, 1977).

While computer programming may not fulfill its prophecy of becoming the new Latin school movement (Mayer, in press) by providing skills that transfer to remote topics, it is likely that its skills transfer to tasks which are similar to programming. Therefore, the similarity of skills used in computer programming and in mathematical problem solving increases the probability of their transfer.

Critical Skills for Problem Solving

Within the integrated problem-solving model, five skills have been identified as critical ingredients for success in problem solving. These are general strategy, planning, logical thinking, variables, and debugging. As illustrated by the description of the problem-solving process, these skills are applicable to problem solving in both mathematics and computer programming. Therefore, it is likely that transfer will occur.

General Strategy

The integrated problem-solving model is a representation of a general strategy for problem solving. This general strategy has been shown to be illustrative of

the process which drives both mathematical problem solving and computer programming. This control process itself must be considered in the sense of a unit of strategic knowledge in human memory. It is a procedural schema that is accessed whenever a problem is encountered. Just as any other representation in long term memory, this executive control schema must be encoded. It is improved by numerous connections to various problems, which are encoded as it is used.

The more information that is related to the executive control schema, the more useful it will be. Therefore, the more problem-solving experience a person has and the more varied it is, the better he will be at solving problems (Kilpatrick, 1978; Lester, 1980; Mayer, 1976). Since computer programming involves a slightly different implementation of the general process, it provides additional connections to the executive control schema. This practice improves the accessibility of the procedural general process in long term memory and, thus, improves problem solving in general.

Planning

Within the problem-solving model, planning is an area of major emphasis. In computer programming, planning is an integral part of the activity, often being formally accomplished with a flow chart. The use of a flow chart is

also appropriate and desirable for students attempting to solve mathematics problems. In using a flow chart the problem solver must make each part of his proposed solution explicit. The operations, the variables and the order and relationship of each must be clearly specified. This requires that the planning process be an integral part of problem solving (Lepper, 1985; Nickerson, 1982).

Another aspect of planning that is appropriate for both computer programming and problem solving is top-down modular design. As mentioned earlier, programs are often designed in a structured manner so that the problem is broken down into successively smaller subproblems. The final design is represented with a structure chart, which illustrates the top-down arrangement. In highly structured languages, such as Pascal and Logo, the subproblems are solved with procedures which are combined into modules which are combined with the main program. A similar process in structured BASIC uses subroutines to solve the subproblems. This design strategy is essential in solving large programming problems. While it is not a requirement in all mathematical problem solving, it can be used with large mathematics problems with very positive results. Using subgoals is a problem-solving strategy suggested by both Newell and Simon (1972) and Polya (1962).

Logical Thinking

Development of logical thinking is a desired goal for all students. However, this goal is not well defined and not well achieved. A recent conference of mathematics educators concluded that even though geometry courses are meant to teach logic, they do not achieve that goal (Fey, 1984). Only a few secondary students complete courses in formal logic, which are often perceived as highly abstract courses.

Computer programming offers a concrete representation of logical forms. One of the most common structures in many computer languages is the IF...THEN...ELSE... statement. This requires the student to use conditional thinking. In programming, this use of conditionals is extensive and the programmer has experience in concrete representation of this concept (Mayer, 1985). This conditional reasoning is a useful skill in mathematical problem solving. A common representation of human long-term memory is that it is structured as a production system in the form of conditionals.

Variables

Since both advanced mathematics and computer programming make extensive use of variables, they both require the student to have skill in understanding and in using variables. Studies have found that many mathematics

students have difficulty in understanding variables (Clement, 1982; Clement, Lochhead, & Monk, 1981). Computer programming offers a similar implementation of variables in a more concrete format (Mayer, 1985; Soloway, Lochhead, & Clement, 1982). The use of variables in programming may be similar to mathematics usage (A = 3 + C), or it may be different (N = N + 1). However, programmers must use variables extensively and in various ways, and, thus, acquire a familiarity with them that is usually not achieved in a mathematics context alone.

Debugging

A final critical skill for problem solving is debugging. As mentioned earlier, there are two types of debugging which are involved in programming. First, any syntactic error in code is detected by the interpreter or the compiler, and an appropriate error message is generated. Thus, the problem solver gets immediate feedback, and many errors are corrected at an early stage. Even though this immediate external feedback is not usually possible in mathematical problem solving, practice would help the student to acquire the habit of making syntactic checks himself.

The other more metacognitive (Flavell, 1979) type of debugging is checking a program which appears to execute properly but gives an incorrect result. This is called a

logic error and is a part of the looking back step of problem solving. The unique contribution of programming to this activity is the concrete result which is obtained quickly and easily. This encourages hypothetical thinking, where the student investigates "What if?" questions (Lepper, 1985; Nickerson, 1982). In mathematical problem solving, this endeavor is considerably more time consuming and more often neglected (Paige & Simon, 1966). Performance of this activity in the programming context would make the student more aware of its value, and more willing to incorporate it into mathematical problem solving.

Summary

Both mathematical problem solving and computer programming have been described in the context of an integrated problem-solving model. These two activities have been shown to be similar in several ways. It is hypothesized that computer programming provides experiences that develop critical problem-solving skills which will transfer to mathematical problem solving. These include (1) a general problem-solving strategy, (2) planning skills, (3) logical thinking skills, (4) understanding of algebraic variables, and (5) debugging skills. Therefore, experience in computer programming

should have a positive effect on mathematical problem solving ability.

Since the question at hand is not how, but to what extent computer programming influences mathematical problem solving, no attempt will be made at this time to identify the particular skills that may transfer. Rather, the cumulative effect of computer programming experience based on a combination of the above five skills will be investigated.

The question which this study will seek to answer is:

Does computer programming experience affect the mathematical problem-solving achievement of high school students?

CHAPTER 2

LITERATURE REVIEW

Potential Background Variables for Computer Programming and Mathematical Problem Solving

In studies seeking to identify prerequisites, general requirements for success in mathematical problem solving have been identified. These are general ability, including both verbal and mathematical (Lester, 1980; Webb, 1979), general knowledge about a variety of topics (Resnick & Ford, 1981; Schoenfeld, 1982), and mathematical background experience (Kantowski, 1977; Lester, 1980; Suydam, 1980; Webb, 1979). Based on these representative studies, the requirements for mathematical problem solving can be summarized as experience (general and mathematical) and ability (verbal and quantitative).

In studies of prerequisites for programming success, the first requirement mentioned is usually general ability. The ACCCEL Project, studying middle school students, found that general ability is a major determinant of degree of success in programming (Dalbey & Linn, 1985; Linn, 1985; Linn & Dalbey, 1985; Mandinach, 1986). In a study of college students, Mayer et al. (1986) identified general

cognitive skill as a good predictor of success in programming. Webb (1985) found that mathematics and verbal ability scores were significantly related to programming knowledge scores for middle school subjects in group and individual classes.

Another potential background variable is gender. The question of gender differences in mathematics is a continuing debate. Briefly, there are differences; males perform better than females in mathematics. The debate is currently centered on the cause of the difference. Some researchers maintain that there is a genetic difference (Benbow & Stanley, 1980; Denno, 1982). Others attribute the difference to results of social and cultural attitudes (Becker, 1981; Burton, 1978; Fennema & Carpenter, 1981; Fennema & Sherman, 1978). Another view is that there is an intricate interaction between gender, ability, and attitudes which affects the process of mathematics achievement (Ethington & Wolfle, 1986).

Gender differences in computer education have also been studied. It is generally believed that computers fall under the same area as mathematics and reflect similar sex differences (Hawkins, 1984). Studies by Chen (1986) and Lockheed (1985) found that males had more positive attitudes toward computers, were more likely to select a computer course, and were more likely to use a computer outside of school. In one study which examined

achievement, Dalbey and Linn (1986) found that females performed as well as or better than males in programming courses. While there is little evidence explicitly relating gender and computer programming, there is considerable evidence that mathematics achievement is related to gender. Therefore, because of the possible relationship between mathematics and computer programming, it is likely that gender is also related to computer programming achievement.

Two other related background variables are socio-economic status and access to a home computer. Petersen and Howe (1979) found that Father's Occupation was significantly related to success in a computer programming course. Wileman, Konvalina, and Stephens (1981) found that the number of hours working on a computer per week was related to programming course grade. Dalbey and Linn (1985) report mixed results on the contribution of out-of-school access to computers to programming achievement. Their results indicate a significant effect for out-of-school access for students in schools which provided "minimum" access, and a non-significant effect for students in schools which provided "efficient and extensive" access.

Several studies of college undergraduate students in computer science courses have found that ability scores are the best predictors of success in programming courses. Some studies identify both mathematical and verbal ability

as important (Leeper & Silver, 1982; Petersen & Howe, 1979; Sauter, 1986; Wileman, Konvalina, & Stephens, 1981). Other studies have found that mathematics ability alone is the best predictor of programming success (Mazlack, 1980; Szymezuk & Frerichs, 1985).

In summary, the most important background variable for both computer programming and mathematical problem solving has been consistently identified as ability. The evidence indicates that both verbal and mathematics ability are involved. Also, experience, especially mathematics experience, has been identified as related to success in problem solving in both computer programming and in mathematics. Research indicates that gender may account for some differences in mathematics achievement. Because mathematics and computer programming seem to be related, it is also likely that gender may account for differences in achievement in computer programming. Also, there is some evidence that socioeconomic factors and access to a home computer may be related to programming success. Again because of the hypothesized close relationship between programming and mathematics, it is likely that socioeconomic factors and access to a home computer may have similar effects on mathematics.

Literature Relating Critical Skills for Problem Solving in Mathematics and in Computer Programming

General Strategy

Some studies to be reviewed in this section have found general evidence that experience in computer programming has a positive effect on problem solving in other contexts. These results suggest that general problem-solving strategy transfers from one activity to the other. There is little explicit evidence of this transfer, but several researchers have hypothesized that this is the underlying cause for relationships that they have observed.

In comparing computer programming and other problem-solving activities, the point is often made that computer programming is an excellent practice medium for problem solving. For example, Nickerson (1982) stated that "the assumption is that programming is prototypical of many cognitively demanding tasks" (p. 42).

In studying the overall relationship of problem-solving ability and computer programming performance, Nowaczyk (1984) constructed a seven-item test of general problem-solving skill. His test included items dealing with logical operations, algebraic solutions, transformations, and identification of mathematical relationships. He compared the score on this problem-solving measure with final grades in computer courses for

301 college undergraduate subjects. He found that the problem-solving score was related to students' programming performances.

Despite the fact that his problem-solving test was questionable because it was short and he presented no reliability or validity statistics, the results of this study offer important tentative evidence. Nowaczyk's conclusion was that there is a positive relationship between problem-solving ability and computer programming. Even though this study did not deal with causality, it can be viewed as justification for further investigation of the relationship.

Using an adaptation of the Nowaczyk instrument as a measure of general problem solving, McCoy and Orey (in press) studied 120 middle and high school students enrolled in BASIC courses. Their results revealed that after one semester of BASIC instruction, general problem-solving scores had significantly increased. This study did not include a control group for comparison, and thus the cause of the increased problem-solving scores is suspect. However, the results indicate a possible effect, and should be further investigated.

Reed and Palumbo (in press) studied 23 college students enrolled in a seven-week BASIC course. Following the BASIC instruction, they found that these students had statistically significant gains in problem solving, as

measured by an instrument consisting of sections from the Ross Test of Higher Cognitive Processes and the Watson-Glaser Critical Thinking Appraisal. These results were replicated in a similar study using 21 students in BASIC and Logo courses (Reed, Palumbo, & Stolar, in press). Both of these studies used the same problem-solving measure, for which no reliability or validity data are presented. Neither study included a control group, and both used intact classes, which were quite small. All these limitations cast doubt on the validity of the results, so further study is needed to determine the effect of programming instruction on problem-solving performance.

Using longitudinal data of mathematically talented college students, Johnson and Harding (1979) found that those students who completed elective computer courses scored higher on a departmental mathematics examination than those who did not elect the courses. This study involved mathematics majors at Cambridge University who were required to pass a yearly Tripos examination at the end of each year of their 3-year course of study. This examination is an extensive and rigorous measure of mathematical problem-solving ability. Students were allowed to choose the computer course as one of their electives. To control for this self-selection, the researchers used the first-year Tripos score (before computer courses were available) to adjust the posttest

score. Regression analysis was then used. The researchers concluded that the computer course had a significant positive effect on the mathematical problem-solving ability of these mathematically talented students. The generalizability of these results is limited because the subjects all had superior mathematics ability scores prior to the study.

In a study of the processes of problem solving, Foster (1972) examined the effect of instruction in use of the computer and flow charts for solving problems on problem-solving performance. His subjects were 68 eighth grade algebra students who were randomly assigned to one of four instructional groups: flow charts, programming in BASIC, both flow charts and programming, or neither flow charts nor programming. At the end of the 12 week treatment, all subjects were tested on the Problem Solving Abilities Test, a 24 item pencil and paper test that measures selected problem-solving processes.

Foster (1972) reported a difference between the computer group and the neither computer nor flow chart group. He found that the computer group had significantly higher total scores on the post test (Problem Solving Abilities Test). He also reported higher scores for the computer group on each of the nine subtests: specifying conditions, selecting a solution, proposing a hypothesis, identifying a pattern, supplying missing information,

selecting relevant data, using a constructed algorithm, correcting an error in a constructed algorithm, and constructing an algorithm. He found no significant differences in performance involving the other two groups, flow charts or both computer and flow charts. His conclusion was that programming the computer positively influences development of the selected problem-solving behaviors measured by his test.

Wells (1981) also studied the processes involved in the activities of computer programming and mathematical problem solving. Her subjects were 15 high school students. She had them talk aloud as they solved 12 mathematics problems and 5 programming problems. She then analyzed the protocols and sought to identify and compare the processes. She found that the same processes were used for both activities. In both computer programming and mathematical problem solving, her subjects used heuristics, looking back, subgoals, trial and error, related problems, persistence, and regular patterns of analysis and synthesis. She also found a significant correlation ($r = 0.77$) between mathematical problem-solving success and computer programming problem-solving success.

In this section, four studies which generally relate problem solving in computer programming and in mathematics have been reviewed. All of these studies found evidence that there is a relationship, and their conclusions were

that the general strategy and/or processes involved in the two activities are similar. Additionally, three studies were reviewed which reported results indicating an increase in problem solving as a result of computer programming experience. This is evidence of the possible causal relationship of computer programming experience on problem-solving ability.

None of these studies goes into any depth in explaining the problem-solving process, but their results are consistent with the Integrated Problem Solving Model presented in Chapter 1. It is reasonable to assume that the problem solvers who were participants in these studies followed the four steps of understanding the problem, devising a plan, carrying out the plan, and looking back. These studies illustrate transfer due to the similarity of the general strategy for problem solving in computer programming and in mathematics.

Planning

The next critical area for problem solving in computer programming and in mathematics is planning. The planning process has been studied in a general sense, and also from the view of using particular programming skills in other types of problem solving.

White and Collins (1983) studied the effect of programming experience on planning. Their subjects were

students age ten to fourteen at a summer computer camp. Each subject was asked to write a plan for a familiar activity (making a peanut butter and jelly sandwich or brushing one's teeth) both before and after programming instruction. The plans were evaluated for completeness. The treatment for the subjects was eight hours of computer instruction which emphasized planning. Post measure scores were significantly greater than pre-measure scores. The researchers concluded that planning, as a component skill of problem solving, can be taught in a computer programming context and that this skill transfers to general problem solving.

The White and Collins (1983) study had several weaknesses. The treatment of eight hours was hardly long enough to expect effects. The two planning tasks were so general that they may have little application within an educational setting. And, the results of the pretest were discussed with the students prior to the post measure. It is likely that this direction had a large influence on the posttest scores. Even with all the problems described above, the results of this study suggest that computer programming experiences may improve planning skills. Since no replication is available, these results are considered weak, but positive, evidence of the effect of computer programming on planning.

Another study of the effect of computer programming experience on planning skill was conducted by Pea and Kurland (1984). Their subjects were children, age 8 - 12, some of whom had completed one year of Logo programming and some with no programming experience. The two groups of children were compared on a planning task, which required them to design a plan for cleaning a classroom. The results showed that there was no difference in planning skill between the programming and nonprogramming groups.

Pea and Kurland (1984) suggested two reasons for their nonsignificant results. First, they state that their task may not have been an appropriate measure of planning skill. The second area of concern, which they considered the probable cause of the lack of planning skill transfer, was the discovery learning pedagogy associated with the Logo instruction given to their treatment group. They seriously questioned this teaching method and suggested that students need "instructional guidance" to develop "advanced thinking strategies" (p. 44).

Project ACCCEL (Dalbey & Linn, 1985; Linn, 1985; Linn & Dalbey, 1985) was a series of studies of the programming behavior of pre-college students. This research was based on a theory of a "Chain of Cognitive Accomplishments" to describe the achievement of programming students. This "chain" consists of three levels: syntactic knowledge of language features, design skills, and problem-solving

skills. According to the theory, in order for students to be able to transfer the design or problem-solving skills, they must move to that point along the "chain."

One of the ACCCEL studies involved using "Spider World," a sublanguage that emphasizes design. Three 12-week middle school programming classes were instructed as follows: (1) using Spider World for three weeks and BASIC for nine weeks, (2) using "Type Attack" (a typing instruction software package) for three weeks and BASIC for nine weeks, and (3) using BASIC for the entire twelve weeks. After the treatment, the Spider World group had better design skills (Dalbey & Linn, 1986).

One problem with this study is the appropriateness of the instrument used to measure design skills. It contained items which were similar to the Spider World activities. Thus, the higher scores of the Spider World group may reflect their familiarity with the format and the material rather than better design skills.

Widespread use of the sublanguage may or may not be feasible; however, it should be noted that there was instruction particularly directed toward design, and it was shown to be effective. It has not been established whether most secondary programming courses emphasize design and problem solving, or whether they concentrate on language features. The ACCCEL studies found wide variation

in the nature of the courses in the schools they observed (Dalbey & Linn, 1985).

Another of the ACCCEL studies was also concerned with planning. Instruction was given to students in the use of a structure diagram to facilitate planning. The researchers observed that those students who had been taught to use a structure diagram were better at planning (Dalbey, Tourniaire, & Linn, 1986).

Thus, there is evidence that computer programming experience may have a positive effect on planning and design skills. One study, even though it had methodological flaws, provided at least weak support for the theory that planning activities associated with programming may improve general planning (White & Collins, 1983). While non-structured discovery lessons were found not to improve planning skills (Pea & Kurland, 1984), other studies have found that explicit instruction in planning and design has a positive effect on planning skills, with a planning language (Dalbey & Linn, 1986) or with the use of structure charts (Dalbey, et al., 1986).

Logical Thinking

Even though logical thinking is important and desirable, it is one area which is extremely difficult to define and measure. Only two studies were available that

attempted to assess the influence of computer programming on logical thinking.

One study examined changes in logical thinking as a result of Logo programming activities (Gorman & Bourne, 1983). For one school year, third grade students were given Logo instruction and then allowed to experiment at the computer either 0.5 or 1 hour per week. On a post measure consisting of slides of combinations of color, shape, size, and number, students were to determine the rule for inclusion based on a conditional (IF...THEN...) relationship. Results showed that the 1 hour group made significantly fewer errors than the 0.5 hour group. This is presented as evidence of improvement in logical thinking.

Similarly, one of the items on the general problem-solving test developed by Nowaczyk (1984) also dealt with the conditional rule. In this case the rule was "IF a letter is sealed THEN it must have a 20 cent stamp on it" (p. 154). The results revealed that the students who got this item correct had significantly higher programming grades than those who missed it.

Even with the inherent difficulties in studying logical thinking, two studies have found reason to conclude that computer programming is positively related to logical thinking. It seems possible that there is a large influence which has not been effectively measured.

Variables

The critical skill area with the most empirical evidence of a transfer of skills from computer programming to mathematical problem solving is variables. This is due, at least in part, to the fact that programming languages use algebraic variables in a manner similar to their mathematical use, but give them an "operational" context. Given the fact that many mathematics students have difficulty with algebraic variables, it may be that this area is the most important short-term effect of computer programming instruction.

In an informal study, Hart (1982) observed that children who had previous computer programming courses performed better on algebra tests. His explanation was that programming helps student to see "letters as labels of stores whose contents can vary" (p. 52).

Soloway, Lochhead, and Clement (1982) studied the relationship of computer programming and the use of variables. Previous studies had found that college engineering students had difficulty in correctly representing relationships in algebraic equations (Clement, 1982; Clement et al., 1981). Soloway et al. (1982) designed two experiments which sought to find out whether students would have similar difficulties with computer programs and with algebra equations in this area. In the first experiment, two groups of college computer students

were given the same problem, but were asked to express it as either an equation or a program. Significantly more subjects in the programming condition were able to express it correctly. In the second experiment, two groups of college subjects were asked to "write a sentence in English" to express the relationship given in both an equation and a program. The number of students who got the equation correct and the program incorrect was significantly less than the number of students who got the program correct and the equation incorrect. The conclusion was that computer programs facilitate the understanding of algebraic variables.

Oprea (1985) studied the effects of programming instruction on generalization and understanding of variable. Her sample was 3 intact groups of sixth graders who received six weeks of programming instruction. The groups were (1) wholistic (taught at the whole program level for mathematically relevant problems, (2) elemental (taught the individual commands necessary to write a complete program), and (3)control (no programming instruction). She developed three instruments for measuring programming ability, generalization, and understanding of variables. Using analysis of covariance to control for pretest scores, she found that both of the programming groups scored significantly higher on all three measures than the control group. She concluded that

programming enhances generalization and understanding of variable. There was no effect for the different instructional methods.

McCoy (1986) studied 21 students, ages 10 to 17, at a summer computer camp. At the end of a 2-week intensive program of BASIC instruction, she found that they showed a significant improvement in ability to use mathematical variables and mathematical problem solving. These gains were measured pre and post by subtests of the Algebra Readiness Test. Again there was no control group, but the results are an indication of a possible effect.

Mayer (1975, 1976, 1979, 1985) explained computer operation in terms of transactions. A transaction is an event in the computer that involves some operation on some object at some location. He proposed that this concrete representation would make programmers better able to understand and use algebraic variables. He verified this idea in a subsequent study (Mayer et al., 1986), where experienced programmers were found to have a better understanding of and facility with mathematical variables. He studied college students in a beginning BASIC course. When he compared them with a control group, he found that the BASIC students gained significantly more in word problem translation, word problem solution, and procedure comprehension. He concluded that all of these specific skills are components of general problem solving, and

therefore, BASIC instruction improves problem solving.

The above studies provide evidence that computer programming improves understanding and skill in algebraic variables. In studies of young children (Hart, 1982), secondary school students (McCoy, 1986; Oprea, 1985), and college students (Soloway et al., 1982; Mayer et al., 1986), experience in computer programming was found to have a positive effect on the knowledge of algebraic variables.

Debugging

The term "debugging" is peculiar to computer programming, but the concept is not. In any kind of problem solving, the solution is checked and, if found not to be acceptable, is re-evaluated. Collins and White (1984) refer to this evaluative function of debugging as critical comprehension. This is the fourth step in the Problem Solving Model, looking back.

Even though debugging is an essential component in problem solving, it is not always considered as a separate skill area; it is more usually thought of as a component of other skills. In a study mentioned earlier (Soloway et al., 1982), one conclusion was that the "better" understanding of variables for programming students was partially due to experience in debugging. While students do not "run equations" in mathematics, they can and should perform actual number testing. This debugging activity

would be useful in mathematics, just as it is in programming.

Two studies have considered debugging as a distinct skill that can be developed and improved. Both of these studies found a positive effect on debugging from computer programming.

Clements and Gullo (1984) concluded that programming enhances thinking skills. They studied 18 six-year-olds. Half the children had Logo programming for 12 weeks, while the other half had 12 weeks of Computer-Assisted-Instruction. The results showed that the Logo group scored higher on measures of reflectivity, divergent thinking, and metacognitive ability. In this study, reflectivity is a more general label used for debugging. The researchers' explanation of these results was that programming activity makes the thinking process conscious, and so the programming children were better thinkers.

In a part of the ACCCEL Project, programming students were observed to be superior to non-programmers in debugging (Dalbey & Linn, 1985). The researchers in this study concluded that debugging is "a generalized skill which can be applied outside the domain of programming" (p. 254). They further stated that the computer programming environment is an ideal place to learn debugging because of the immediate feedback.

Summary

This chapter has examined the available literature on the critical skills involved in computer programming and mathematical problem solving. While there have not been a large number of studies in this relatively young field, the available results provide important information.

The first area of research examined background variables for computer programming and/or mathematical problem solving. The results of several studies identified ability, both verbal and quantitative, as a necessary prerequisite for both computer programming and mathematical problem solving. In addition, experience, particularly in mathematics, was identified as a significant variable in predicting success in computer programming and/or mathematical problem solving. Additionally, gender, socio-economic status, and access to a home computer are possible causal influences on mathematical problem solving and computer programming.

Despite the small number of studies relating problem solving in computer programming and mathematics, the results are quite consistent. There is variation in research methods, in subjects, in treatment, and in the particular aspect of problem solving studied. In each of the five areas identified as critical for problem solving, studies have found small and tentative, but positive,

indications that computer programming may have an effect on mathematical problem solving.

Studies of the overall relationship between computer programming and mathematical problem solving have found that they are significantly related and that experience in programming causes improvement in mathematical problem solving. Other studies examined the general process of problem solving and concluded that computer programming and mathematical problem solving involve similar processes and that these processes are improved by computer programming experiences.

The research on planning is somewhat difficult to integrate. One study found that computer programming improved general planning (White & Collins, 1983), and a similar study found that computer programming experience, in a discovery instructional mode, did not improve planning (Pea & Kurland, 1984). While these two studies seem contradictory, an important difference in the treatment might explain their different results. The White and Collins study involved structured instruction which emphasized planning, and the Pea and Kurland treatment was unstructured discovery. This treatment variation is further illustrated in the ACCCEL studies, where students who were explicitly taught planning skills demonstrated better planning.

Logical thinking skill, as an outcome of programming, is an intuitively likely possibility. However, the difficulty in defining and measuring logical thinking makes research in this area difficult. Two studies have found evidence that computer programming students score higher on measures of one area of logical thinking, conditional reasoning.

Several studies have found that computer programming experience has a positive effect on students' ability to understand and use algebraic variables. There is more evidence in this area, probably because it is easier to measure.

The last area of possible transfer is debugging. Two studies found that programmers were better at general debugging, or looking back, skills.

Thus, to summarize the small number of research studies in this area, there is evidence that computer programming has positive effects in several critical skill areas which are also common to mathematical problem solving. Results have indicated that computer programming improves general problem-solving strategy, planning, logical thinking, algebraic variables, and debugging. Therefore, because of improvement in any or all of the above five areas, computer programming experience should improve mathematical problem-solving ability.

There has been little research into the relationship of computer programming and mathematical problem solving. As pointed out in previous paragraphs, most studies in this area have serious methodological flaws. Because of the general weakness of research in this area, there is a need for reliable investigation of this possible relationship.

CHAPTER 3

METHODOLOGY

There is evidence, as reviewed in Chapter 2, to indicate that computer programming experience may cause an improvement in mathematical problem-solving ability. An appropriate statistical technique for testing this theory is path analysis.

Path Analysis Procedures

Path analysis is a method for studying the direct and indirect effects of variables hypothesized as causes of variables treated as effects. This method is applied to a causal model formulated by the researcher on the basis of related research and theory. A path diagram is constructed to display graphically the hypothesized pattern of causal relations among the variables. Once a researcher has constructed a causal model of explicit theoretical relationships, data is gathered to provide evidence as to the tenability of the theory.

The variables in the path analysis model are distinguished as exogenous or endogenous. An exogenous variable is a variable whose causes are considered to be

outside the model. Exogenous variables may be related to each other or to other non-specified variables, but the model does not deal with these questions. An endogenous variable is one which is inside the model and whose variability is explained by exogenous or other endogenous variables in the system.

A recursive[1] path model is one in which the causal flow is unidirectional. The associations in this type of model can be decomposed to allow examination of direct and indirect effects. The direct effects are obtained by regressing each endogenous variable on all of its antecedent variables. The standardized regression coefficients (beta weights) are the path coefficients. The indirect effects are the products of the combinations of path coefficients.

---

[1]NOTE: The term "recursive" as used to describe a unidirectional path model is not consistent with its earlier use to describe a problem-solving model that has repeating feedback loops. As used here, the definition of "recursive" is drawn from the statistics literature. The use of "recursive" at other places in this document has been drawn from the more general mathematics and computer programming literature.

Description of the Path Model for Mathematical Problem Solving

The path model hypothesized in this study had three exogenous variables (ability, socioeconomic status, and gender) and four endogenous variables (mathematics experience, computer programming experience, access to a home computer, and mathematical problem-solving achievement). See Figure 2.

This model is supported by the literature, as reviewed in Chapter 2. Computer programming experience has been identified as significantly related to mathematical problem solving. This analysis is designed to test the theory that this relationship is causal, and that computer programming experience causes a change in mathematical problem-solving performance. In order to control for the other variables identified as associated with both computer programming and mathematical problem solving, ability, socioeconomic status, gender, mathematics background, and access to a home computer were also included in the model. Therefore, this model was hypothesized to be theoretically and logically appropriate to explain mathematical problem-solving ability.

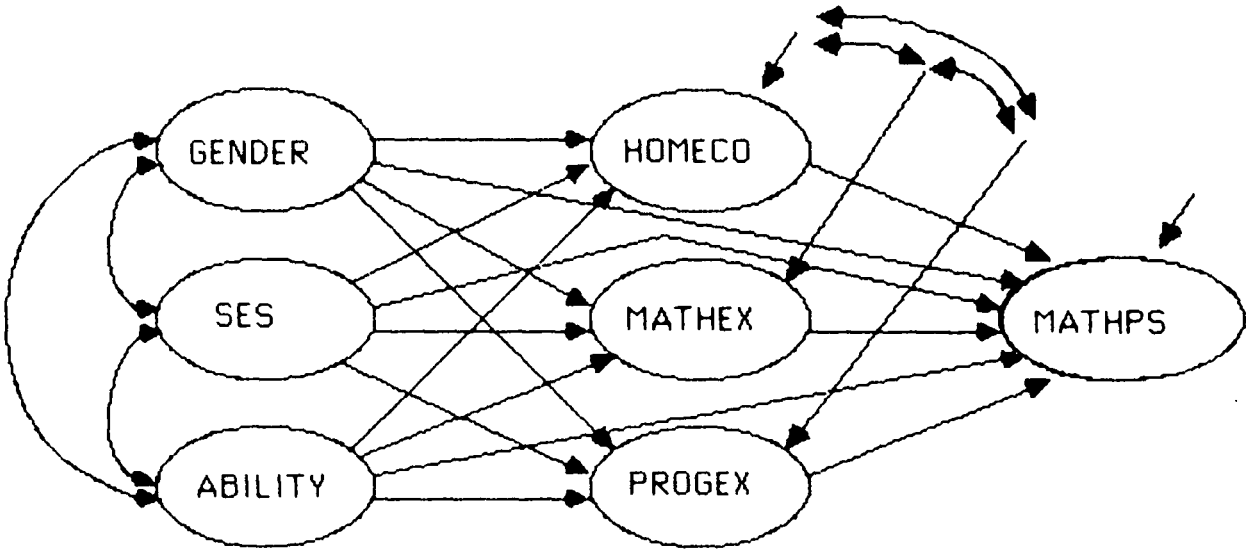

Figure 2. Path Model of Mathematical Problem Solving

NOTES: MATHPS = mathematics problem solving achievement,
PROGEX = years of computer programming experience,
MATHEX = years of higher mathematics experience,
HOMECO = access to a home computer,
ABILITY = general ability,
SES = socio-economic status,
GENDER = gender of participants.

Subjects

Subjects for this study were 800 high school seniors attending seven public high schools in seven different school districts in southwest Virginia. The selection of the schools was based on convenience and accessibility. However, it was assumed that students in these schools were not unique or unlike the general population. They included a mixture of urban and rural areas and various socio-economic levels. School sizes varied from very small to large. In the selected schools, all seniors in the school were included in the sample.

Variables

The study included seven variables. These were operationally defined as follows.

Mathematics problem-solving achievement was defined as the score on Level H of the Mathematics Problem Solving subtest of the Science Research Associates Achievement Series. This test is a part of the test battery given to all Virginia high school students in the spring of the junior year. This subtest consists of 40 multiple choice items and has a reported reliability of 0.91. Raw scores were obtained from student records.

Computer programming experience was defined as the number of computer programming courses the student had completed at the end of the junior year. Summer computer camp experience counted as one semester. Students were asked to give this information in the Student Questionnaire.

Mathematics experience was defined as the number of years of higher mathematics courses the student had completed at the end of the junior year. Higher mathematics courses were defined as algebra, geometry, trigonometry, calculus, etc. Students were asked to give this information on the Student Questionnaire.

Access to a home computer was determined from a question on the Student Questionnaire. Students were asked to check "Yes" (coded as 1) or "No" (coded as 0).

Ability was defined as the sum of the Numerical Ability and Verbal Reasoning scores on the Differential Aptitude Tests. The test manual claims that a combination of these two scores is a valid measure of general ability. This test had been given to all students in their ninth grade year as a part of the Virginia State Testing Program. It has reported reliabilities in the low 0.90's. Raw scores were obtained from student records.

Socioeconomic status was assessed by items on the Student Questionnaire. Students were asked to mark a checklist with the occupation of their father and their

mother, and a checklist with the education level of both parents. These lists were coded according to the socio-economic indices presented by Duncan (1961). The mother's occupation was not used because the scales are intended for use with the employed labor force. A large number of unemployed housewives whould make the scale unreliable. The three remaining items were then converted to standard scores and summed to form one socioeconomic index.

Gender was determined from a question on the Student Questionnaire. Answers to the question were coded female = 0, and male = 1.

Data Collection

First, permission to conduct the study was obtained from county superintendents. Then the researcher visited each school and discussed the study with principals and then with teachers. Permission slips/student questionnaires for all seniors were distributed to teachers (See Appendix A). The choice of the class for the distribution of the questionnaire was left to the principal; all schools used English or government classes.

Approximately two weeks later, the researcher again visited the school and picked up the forms from the teachers. Then the researcher examined participants' records and obtained test scores for the Differential

Aptitude Tests and the Mathematical Problem Solving subtest of the Science Research Associates Achievement Series.

Data Analysis

Data analysis consisted of calculation of the multiple regression equations for the model to obtain path coefficients. These coefficients, indicating direct effects, were evaluated for significance and magnitude. Cases with missing data were deleted on a pairwise basis for the socioeconomic variable. Listwise deletion was used for all other variables. This resulted in a sample size ranging from 593 to 800.

Additionally, the direct associations were decomposed (using the GEMINI program, Wolfle & Ethington, 1985) and the direct and indirect effects were examined.

CHAPTER 4

RESULTS

Descriptive Information

Means and standard deviations of all variables are presented in Table 1. The sample of 800 southwest Virginia high school students was about evenly divided between male and female participants, with 51 percent female and 49 percent male. Table 2 describes the frequencies.

Raw ability scores had a mean of 46.34 and a standard deviation of 16.034. Figure 3 shows that this distribution was approximately normal.

Socioeconomic status had a mean of 0.107, with a range of -3.5 to +6.5 on the summed scale. The standard deviation was 2.508. A frequency histogram (See Figure 4) illustrates that this distribution was positively skewed.

The mean number of years of computer programming experience was 0.386. The frequencies in Table 3 indicate that nearly two-thirds of the students in the sample had no computer programming experience. Of those who did have programming experience, most had only one year, with about 4 percent of the total sample having two or three years.

Table 1

Means and Standard Deviations of All Variables

	Mean	Standard Deviation
GENDER*	.490	.500
SES**	.107	2.508
ABILITY*	46.340	16.034
HOMECO*	.265	.442
MATHEX*	2.129	1.204
PROGEX*	.386	.579
MATHPS*	54.991	18.065

*N=800
**N=593

SES = Socioeconomic status,
ABILITY = general ability,
HOMECO = access to a home computer,
MATHEX = years of higher mathematics experience,
PROGEX = years of computer programming experience,
MATHPS = mathematics problem solving achievement.

Table 2

Frequency Table for Gender

	FREQUENCY	PERCENT
FEMALE	408	51.0
MALE	392	49.0
TOTAL	800	100.0

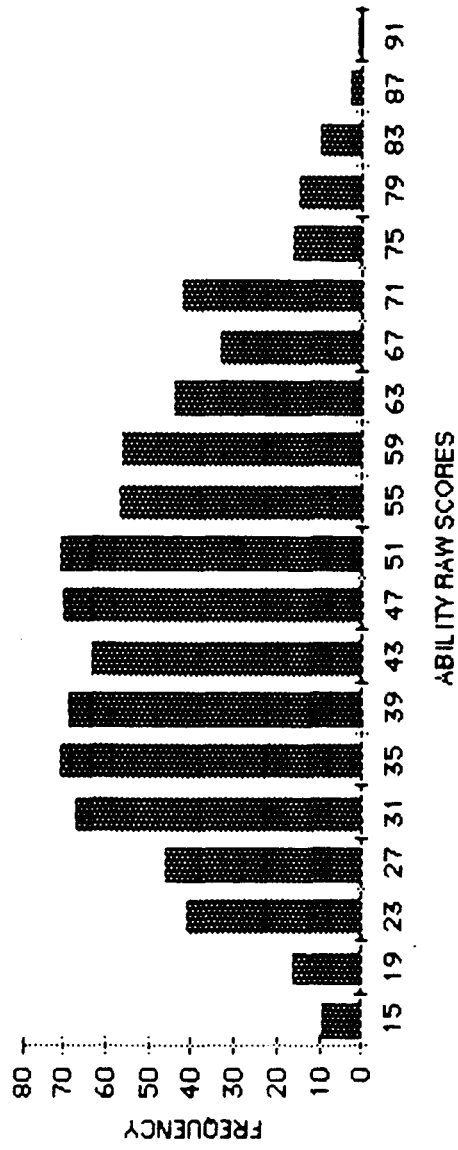

FIGURE 3
FREQUENCY OF ABILITY SCORES
(N = 800)

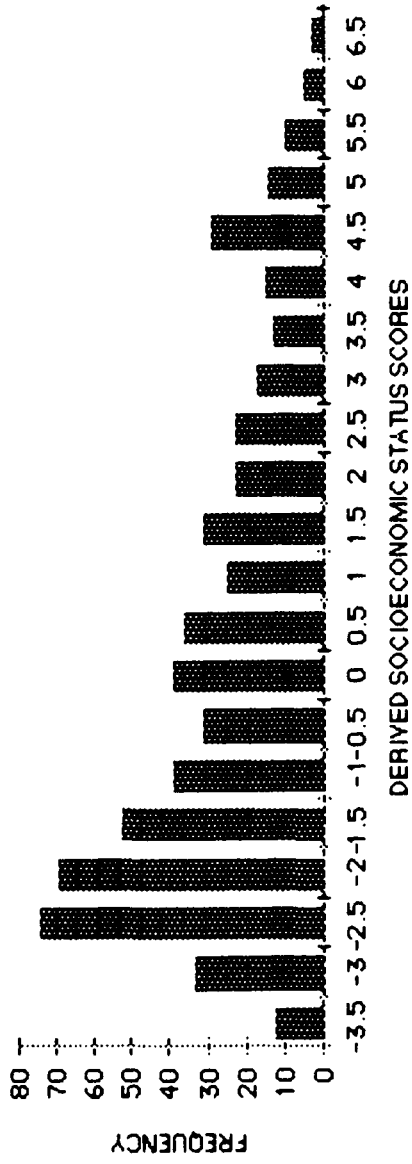

FIGURE 4
FREQUENCY OF SOCIOECONOMIC STATUS SCORES
(N = 593)

Table 3

Frequency Table for Programming Experience

	FREQUENCY	PERCENT
0 YEARS	527	65.9
1 YEAR	240	30.0
2 YEARS	30	3.7
3 YEARS	3	.4
TOTAL	800	100.0

The mean number of years of mathematics experience was 2.129, with a standard deviation of 1.204. Table 4 indicates that over half of the students had three years of mathematics experience, with a few students having four years and the others having zero, one or two years.

Only 26.5 percent of the students had a home computer (See Table 5).

The mean of the ultimate dependent variable, mathematical problem solving, was 54.991, with a standard deviation of 18.065. Figure 5 shows that this distribution appears to be normal.

Zero order correlations were calculated for all combinations of variables. While these statistics will not be discussed, they were used as input data for the regression analyses. The correlation matrix for all variables is included as Appendix B.

Causal Effects on Mathematical Problem Solving

To test the theory of the causal effect of programming experience on mathematical problem solving, a path model was constructed (See Figure 2). Also included were three background variables (ability, socioeconomic status, and gender) and two other endogenous variables (mathematics experience and access to a home computer). This model was

Table 4

Frequency Table for Mathematics Experience

	FREQUENCY	PERCENT
0 YEARS	130	16.2
1 YEAR	111	13.9
2 YEARS	118	14.7
3 YEARS	408	51.0
4 YEARS	33	4.1
TOTAL	800	100.0

Table 5

Frequency Table for Home Computer

	FREQUENCY	PERCENT
NO	588	73.5
YES	212	26.5
TOTAL	800	100.0

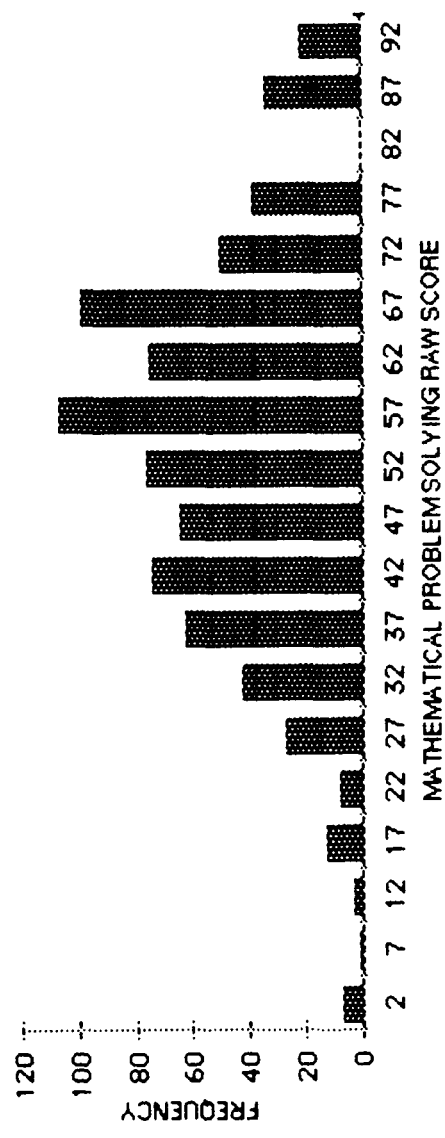

FIGURE 5
FREQUENCY OF MATHEMATICAL PROBLEM SOLVING SCORES
(N = 800)

then tested using the data described above. Standardized path coefficients for direct effects are included in the model presented as Figure 6.

As the model illustrates, the direct effect from ability on mathematical problem solving is quite large (.64). Three path effects are moderate: gender (.17), mathematics experience (0.15), and programming experience (0.08). The other two path coefficients are quite small: access to a home computer (0.01) and socioeconomic status (0.01). The residual path for mathematical problem solving is 0.63.

The results of the multiple regression analysis to predict mathematical problem solving are presented in Table 6. The variables which significantly contribute to the prediction ($p<.01$) are (in order of magnitude of contribution) ability, gender, mathematics experience, and programming experience. The non-significant predictors were access to a home computer and socioeconomic status. The R square for the prediction equation including these six variables was equal to 0.606, indicating that 60.6 percent of the variance in mathematical problem solving was accounted for by these independent variables.

Regression coefficients indicate that each standard deviation of increase in ability score adds 0.639 standard deviations to mathematical problem-solving score. Similarly, for every standard deviation of mathematics

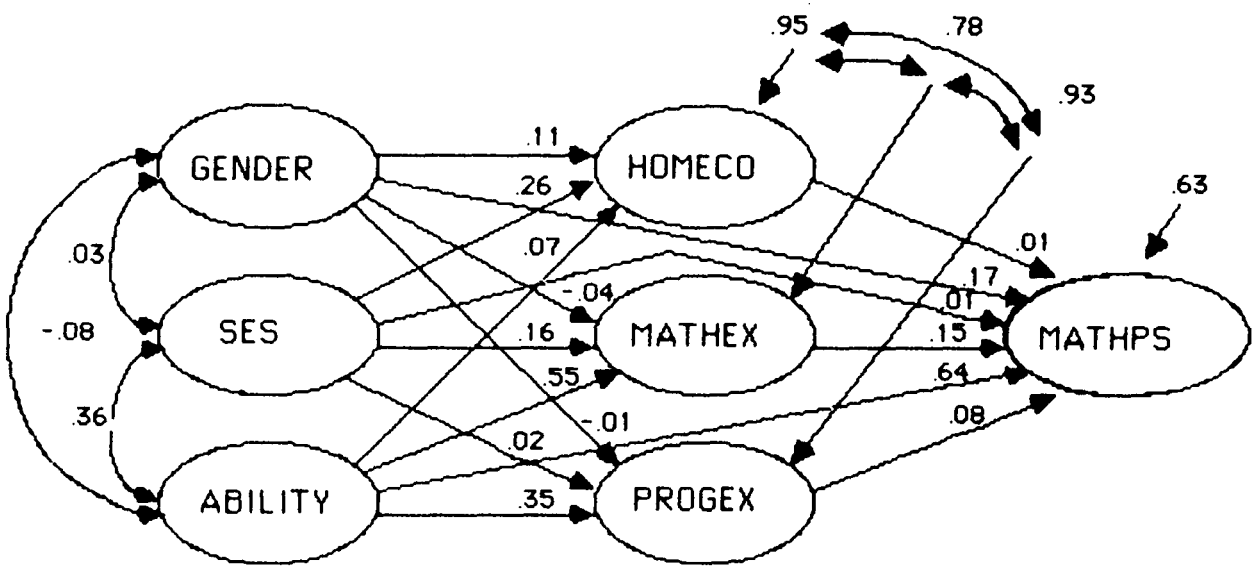

FIGURE 6.

Path Model Explaining Mathematical Problem Solving

NOTES: MATHPS = mathematics problem solving achievement,
PROGEX = years of computer programming experience,
MATHEX = years of higher mathematics experience,
HOMECO = access to a home computer,
ABILITY = general ability,
SES = socio-economic status,
GENDER = gender of participants.

Table 6

Summary of Regression Results
for Mathematics Problem Solving

INDEPENDENT VARIABLES					
PROGEX	MATHEX	HOMECO	ABILITY	SES	GENDER
Standardized Coefficients:					
.075	.147	.009	.639	.010	.166
Regression Coefficients:					
2.339*	2.198*	0.373	0.720*	0.069	5.982*
(.769)	(.439)	(.972)	(.033)	(.180)	(.816)

Constant	R^2
13.001	.606*

Standard errors shown in parentheses
*p<.01
PROGEX = years of computer programming experience,
MATHEX = years of higher mathematics experience,
HOMECO = access to a home computer,
ABILITY = general ability,
SES = socioeconomic status.

experience the mathematical problem-solving score can be expected to increase 0.147 standard deviations. And, for every standard deviation of programming experience the mathematical problem-solving score can be expected to increase 0.075 standard deviations.

Further analysis was performed to decompose the original bivariate correlations into causal effects (direct and indirect) and noncausal covariation. Direct and indirect effects are causal influences from one variable to another in the model, either directly or through another variable (indirect). Noncausal covariation is defined as the influence of prior variables in the model, or of mutually occurring exogenous variables, or both. These decompositions are presented in Table 7. Of particular interest is the relationship of programming experience with mathematical problem solving. The correlation of 0.355 can be broken down to direct effect of 0.075 and noncausal covariation (effects from other variables in the model, primarily ability) of 0.280. Similarly, the correlation of mathematics experience and mathematical problem solving is 0.555, but only 0.147 is a direct causal effect; the other 0.408 is due to non-causal covariation.

Of the background variables, ability has a correlation of 0.746 with mathematical problem solving. Of this, 0.639 is direct causal effect, and 0.107 is indirect causal effect. None of this relationship is due to noncausal

Table 7

Decomposition of Bivariate Correlations with Mathematics Problem Solving

	RELATED VARIABLES					
Type of Relation	PROGEX	MATHEX	HOMECO	ABILITY	SES	GENDER
Original Correlation	.355	.555	.173	.746	.306	.099
Total Causal	.075	.147	.009	.746	.037	.161
Direct	.075	.147	.009	.639	.010	.166
Indirect	.000	.000	.000	.107	.027	-.005
Noncausal Covariation	.280	.408	.164	.000	.269	-.062

Note: "Noncausal covariation" is attributable to prior variables in the model, or to mutually occurring exogenous variables, or both.

PROGEX = years of computer programming experience,
MATHEX = years of higher mathematics experience,
HOMECO = access to a home computer,
ABILITY = general ability,
SES = socioeconomic status.

covariation. Gender has a correlation of 0.099 with mathematical problem solving; 0.166 is direct causal effect, -0.005 is indirect causal effect, and -0.062 is noncausal covariation. Socio-economic status has a correlation of 0.306 with mathematical problem solving; 0.010 is a direct causal effect, 0.027 is the indirect effect, and 0.269 is due to noncausal covariation.

CHAPTER 5

CONCLUSIONS

The purpose of this study was to investigate the question, "Does computer programming experience affect the mathematical problem-solving achievement of high school students?" Evidence to answer this question was sought in the context of a path model for predicting mathematical problem solving. Six causal variables were hypothesized and included in the model. The exogenous (or background) variables were ability, gender, and socioeconomic status. The endogenous variables were mathematics experience, programming experience, and access to a home computer. These were all theorized to be causes of mathematical problem-solving achievement. This model was tested using a sample of 800 high school seniors from seven high schools. Data generally supported the model.

Background Variables and Mathematical Problem Solving

The results indicated that ability had a large causal effect on mathematical problem solving. This influence was expected, due to the overall strong relationship between ability and school achievement in any subject. In addition

to a direct effect of 0.639, ability also had a slight indirect effect of 0.107, mostly through mathematics experience and programming experience.

This influence is consistent with the strong relationship between ability and mathematical problem solving reported by other researchers (Lester, 1980; Webb, 1979). Additionally, the indirect effect of ability was expected because a number of studies have identified a strong relationship between ability and computer programming success (e.g., Dalbey & Linn, 1985; Leeper & Silver, 1982; Mayer et al., 1986; Petersen & Howe, 1979). It is no surprise that students with higher ability had higher achievement regardless of the content area.

Gender was found to have a moderate causal effect on mathematical problem-solving achievement. The path data show a direct effect that being male causes higher mathematical problem-solving achievement. This result is consistent with many other studies that have found that males typically outscore females in mathematics (Benbow & Stanley, 1980; Ethington & Wolfle, 1986) and in computer programming (Lockheed, 1985).

The other background variable, socioeconomic status, had very little causal effect on mathematical problem-solving achievement. This result, contrasting with results of Petersen and Howe (1979), may be due in part to the equality of access practiced by the public schools.

Experience Variables and Mathematical Problem Solving

The three endogenous variables considered as causes of mathematical problem solving were all experience-oriented. First, mathematics experience had a moderate causal effect on mathematical problem solving. It was expected that students who had more mathematics experience would perform better on mathematics measures. Most of the relationship between mathematics experience and mathematical problem solving is noncausal, attributable to ability. Having a home computer had only a very slight causal effect on mathematical problem solving. Only 26.5 percent of the sample reported having a home computer, and no attempt was made to determine the amount of time the student actually used it. Also, similar to results reported by Dalbey and Linn (1985), the schools of this sample provided extensive hands-on experience, suggesting that out-of-school access was a less critical influence.

Computer Programming and Mathematical Problem Solving

The final variable theorized to be causally related to mathematical problem solving was computer programming experience. The direct path coefficient from programming experience to mathematical problem solving is moderate in size, but significant. This result is the basis for

answering the research question: Computer programming does have a causal effect on mathematical problem solving in high school students. Once the effects of ability and gender are controlled, computer programming, along with mathematics experience, is a significant cause of mathematical problem-solving achievement score.

This implies that there is evidence to support the theory of transfer of skills from computer programming to mathematical problem solving. This transfer is possible because of the similarity of the two activities. Both follow the integrated problem-solving model, which includes search and retrieval at four steps: understanding the problem, devising a plan, carrying out the plan, and looking back.

Also, the results can be taken as evidence of transfer of skills common to both tasks. Both computer programming and mathematical problem solving were shown to use similar skills in general strategy, planning, logical thinking, algebraic variables, and debugging. While the present study was not designed to identify which particular skills had transferred, it has shown that some or all of these skills did transfer. Again, this is near transfer, as a result of the similarity of the two tasks. Further research should more closely examine these specific skills in relation to programming experience.

The level of computer programming expertise of the students was not assessed, but due to the fact that only about four percent had two or more years of experience, it is likely that they were at the lower end of Linn's Chain of Cognitive Accomplishments (Linn & Dalbey, 1985). With only one year of experience in computer programming, they are probably at the language features stage. It is unlikely that they would progress to design skills or problem-solving skills in one year. In other words, the "experienced" students in this study with only one year of computer programming experience, could be classified as novice or, at best, intermediate level programmers. Since a significant causal effect was found for computer programming experience at this level, it is quite possible that a much larger effect would be found for a sample with more computer programming experience. Since experts have deeper and more accessible knowledge structures, it seems likely that expert programmers would transfer even more skills than novices or intermediates.

This study was designed to assess the cumulative effect of high school programming experiences. The outcome reflects the current public school computer curricula. Most of the seven high schools in the sample offered only one or two programming courses, and many students did not select them until their senior year. While these conditions are reflective of current practice in the public

schools, further research is necessary to assess the cognitive effect of varied amounts of programming experience.

Implications for Curriculum

The background variables may be considered constant (or nearly so). It is unlikely that general ability and socioeconomic status scores will be changed drastically as a result of school experiences. By definition, gender remains a constant.

Therefore, if an educational goal is to increase mathematical problem-solving achievement scores, then there is evidence here to indicate that in addition to experience in mathematics, experience in computer programming is also effective in causing the mathematical problem-solving score to increase. The two experiences have statistically unique effects, implying that if both computer programming and mathematics experiences are a part of the student's curriculum, then the mathematical problem-solving score can be expected to increase more than it would with only one or the other.

Summary

This study adds support for the theory that computer programming improves problem solving in mathematics. A path model was constructed to show that mathematical problem solving is causally related to ability, gender, socioeconomic status, mathematics experience, computer programming experience, and access to a home computer. This model was tested with a sample of 800 high school students from seven different schools. The data supported the theory, indicating that ability, gender, mathematics experience, and computer programming experience were significant causes of mathematical problem-solving achievement.

Even though this study measured quantity of programming experience rather than quality or programming achievement, it does provide preliminary evidence and a basis for further study of the relationship of computer programming and mathematical problem solving. The results of this study support the theory that increased computer programming experience is a significant cause of increase in mathematical problem-solving achievement score.

REFERENCES

Adelson, B. (1981). Problem solving and the development of abstract categories in programming languages. Memory and Cognition, 9 (4), 422-433.

Battista, M. T. & Steele, K. J. (1984). The effect of computer-assisted and computer programming instruction on the computer literacy of high ability fifth grade students. chool Science and Mathematics, 84 (8), 649-658.

Becker, J. R. (1981). Differential treatment of females and males in mathematics classes. Journal for Research in Mathematics Education, 12 (1), 40-53.

Benbow, C. P. & Stanley, J. C. (1980). Sex differences in mathematical ability: Fact or artifact? Science, 210 (12), 1262-1264.

Bitter, G.G. & Camuse, R. A. (1984). Using a microcomputer in the classroom. Reston, VA: Reston Publishing.

Bork. A. (1985). Personal computers for education. New York: Harper and Row.

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

Burton, G. M. (1978). Why Susie can't--or doesn't want to--add. In J. E. Jacobs (Ed.). Perspectives on women and mathematics (pp. 35-55). Columbus, OH: ERIC.

Carpenter, T. P., Corbitt, M. K., Kepner, H. S., Lindquist, M. M., & Reys, R. E. (1980). NAEP note: Problem solving. Mathematics Teacher, 73 (9), 427-433.

Chase, W. C. & Simon, H. A. (1973). Perception in chess. Cognitive Psychology, 4, 55-81.

Chen, M. (1986). Gender and computers: The beneficial effects of experience on attitudes. Journal of Educational Computing Research, 2 (3), 265-282.

Clement, J. (1982). Algebra word problem solutions: Thought processes underlying a common misconception. Journal for Research in Mathematics Education, 13 (1), 16-30.

Clement, J., Lochhead, J., & Monk, G. S. (1981). Translation difficulties in learning mathematics. American Mathematical Monthly, 88, 286-290.

Clements, D. H. & Gullo, D. F. (1984). Effects of computer programming on young children's cognition. Journal of Educational Psychology, 76 (6), 1051-1058.

Collins, R. W. & White, K. B. (1984). An investigation of componential skill relationships in programming. AEDS Journal, 17, 123-129.

Dalbey, J. & Linn, M. C. (1985). The demands and requirements of computer programming: A literature review. Journal of Educational Computing Research, 1 (3), 253-274.

Dalbey, J., & Linn, M. C. (1986). Cognitive consequences of programming: Augmentations to BASIC instruction. Journal of Educational Computing Research, 2 (1), 75-93.

Dalbey, J., Tourniaire, F., & Linn, M. C. (1986). Making programming instruction cognitively demanding: An intervention study. Journal of Research in Science Teaching, 23 (5), 427-436.

DeGroot, A. D. (1965). Thought and choice in chess. The Hague: Mouton.

Denno, D. (1982). Sex differences in cognition: A review and critique of the longitudinal evidence. Adolescence, 16 (68, 779-787.

Duncan, O.D. (1961). A socioeconomic index for all occupations. In A. J. Reiss, Jr. (Ed.), Occupations and Social Status. New York: The Free Press of Glencoe.

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

Ethington, C. A. & Wolfle, L. M. (1986). A structural model of mathematics achievement for men and women. American Educational Research Journal, 23 (1), 65-75.

Fennema, E. & Carpenter, T. P. (1981). Sex-related differences in mathematics: Results from national assessment. Mathematics Teacher, 74 (10), 554-559.

Fennema, E. H. & Sherman, J. A. (1978). Sex-related differences in mathematics achievement and related factors: A further study. Journal for Research in Mathematics Education, 9 (5),189-203.

Fey, J. (Ed.) (1984). Computing and mathematics: Impact on secondary school curricula. Reston, VA: National Council of Teachers of Mathematics.

Flavell, J. H. (1979). Metacognition and cognitive monitoring. American Psychologist, 34, 906-911.

Foster, T. E. (1972). The effects of computer program experiences on student problem behaviors in eighth grade mathematics. Doctoral dissertation, University of Wisconsin, Madison.

Frederiksen, N. (1984). Implications of cognitive theory for instruction in problem solving. Review of Educational Research, 54 (3), 363-407.

Gagne, R. M. (1983). Some issues in the psychology of mathematics instruction. Journal for Research in Mathematics Education, 14 (1), 7-18.

Gorman, H. & Bourne, L. E. (1983). Learning to think by learning LOGO: Rule learning in third-grade computer programmers. Bulletin of the Psychonomic Society, 21 (3), 165-167.

Hart, M. (1982). Using computers to understand mathematics. Mathematics Teaching, 52-54.

Hawkins, J. (1984). Computers and girls: Rethinking the issues (Tech. Rep. No. 24). New York: Bank Street College of Education, Center for Children and Technology.

Hinsley, D. A., Hayes, J. R., & Simon, H. A. (1977). From words to equations meaning and representation in algebra word problems. In M. A. Just & P. A. Carpenter (Eds.). Cognitive processes in comprehension (pp. 89-105). Hillsdale, NJ: Erlbaum.

Hudgins, B. B. (1977). Learning and thinking: A primer for teachers. Itasca, IL: Peacock.

Hunter, B., Dearborn, D., & Snyder, B. (1983, October). Computer literacy in the K-8 curruculum. Phi Delta Kappan, 115-118.

Johnson, D. C. & Harding, R. D. (1979). University level computing and mathematical problem-solving ability. Journal for Research in Mathematics Education, 10, 37-55.

Johnston, J. (1985, January). Microtrends in the middle school: Beyond computer-literacy training. Educational Technology, 25, 29-32.

Kantowski, M. G. (1977). Processes involved in mathematical problem solving. Journal for Research in Mathematics Education, 8 (3), 163-180.

Kantowski, M. G. (1982). Problem solving: Searching for solutions. In S. Hill (Ed.). Education in the 80's: Mathematics. Washington, DC: National Education Association.

Kilpatrick, J. (1967). Analyzing the solution of word problems in mathematics: An exploratory study. Doctoral dissertation, Stanford Univerisity, Stanford.

Kilpatrick, J. (1969). Problem solving in mathematics. Review of Educational Research, 39 (4), 523-534.

Kilpatrick, J. (1978). Research on problem-solving in mathematics. School Science and Mathematics, 78 (3), 189-192.

Klatzky, R L. (1980). Human memory: Structures and processes. New York: W. H. Freeman.

Kleiman, G. M. (1984). Brave new schools: How computers can change education. Reston, VA: Reston Publishing.

Kulm, G. (1984). Mathematics processes and computers in the junior high school. Mathematics Teacher, 77, 628-630.

Laney, J. D. (1985, January). Obstacles to the computer literacy ideal: Will there be burnout on the launching pad? Educational Technology, 25, 50-53.

Leeper, R. R. & Silver, J. L. (1982). Predicting success in a first programming course. SIGCSE Bulletin, 14 (1), 147-150.

Lepper, M. R. (1985). Microcomputers in education: Motivational and social issuse. American Psychologist, 40 (1), 1-18.

Lester, F. K. (1980). Research on mathematical problem solving. In R. J. Shumway (Ed.), Research in mathematics education (pp. 286-323). Reston, VA: National Council of Teachers of Mathematics.

Linn, M. C. (1985). The cognitive consequences of programming instruction in classrooms. Educational Researcher, 14 (5), 14-29.

Linn, M. C. & Dalbey, J. (1985). Cognitive consequences of programming instruction: instruction, access, and ability. Educational Psychologist, 20 (4), 191-206.

Lockheed, M. E. (1985). Women, girls, and computers: A first look at the evidence. Sex Roles, 13 (3/4), 115-122.

Lockheed, M. E., & Mandinach, E. B. (1986). Trends in educational computing: Decreasing interest and the changing focus of instruction. Educational Researcher, 15 (5), 21-26.

Luehrmann, A. (1981). Computer literacy--What should it be? Mathematics Teacher, 74, 682-686.

Luehrmann, A. (1984, April). Computer literacy. Electronic Learning, 38-40.

Mandinach, E. B. (1986, April). Aspects of programming courses that foster problem solving. Paper presented at the American Educational Research Association annual meeting, San Francisco.

Mayer, R. E. (1975). Different problem-solving competencies established with and without meaningful models. Journal of Educational Psychology, 67 (6), 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 (2), 143-150.

Mayer, R. E. (1979). A psychology of learning BASIC. Communications of the ACM, 22, 589-593.

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

Mayer, R. E. (1983). Thinking, problem solving, cognition. New York: Freeman.

Mayer, R. E. (1985). Learning in complex domains: A cognitive analysis of computer programming. The Psychology of Learning and Motivation, 19, 89-130.

Mayer, R. E. (In press). The elusive search for teachable aspects of problem solving. In J. Glover & R. Ronning (Eds.), History of educational psychology. New York: Academic Press.

Mayer, R. E., Dyck, J. L., & Vilberg, W. (1986). Learning to program and learning to think: What's the connection? Communications of the ACM, 29 (7), 605-610.

Mazlack, L. J. (1980). Identifying potential to acquire programming skill. Communications of the ACM, 23 (1), 14-17.

McCoy, L. P. (1986, March). Correlates of Computer Programming in Children. Paper presented at Eastern Educational Research Association Annual Meeting, Miami.

McCoy, L. P. & Orey, M. A. (in press). Computer programming and general problem solving by secondary students. Computers in the Schools.

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

Michener, E. R. (1978). Understanding understanding mathematics. Cognitive Science, 2, 361-383.

National Council of Supervisors of Mathematics. (1977). Position paper on basic mathematical skills. Washington, DC: National Institute of Education.

National Council of Teachers of Mathematics. (1980). An agenda for action. Reston, VA: NCTM.

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

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

Nowaczyk, R. H. (1984). The relationship of problem-solving and course performance among novice programmers. International Journal of Man-Machine Studies, 21, 149-160.

Nutter, J. T. & Hassell, J. (1985). Programming schemes in program understanding and maintenance: An introduction. AEDS Journal, 18,195-205.

Oprea, J. M. (1985). Computer programming and mathematical thinking.Proceedings of the seventh annual meeting psychology of mathematics education--North American chapter, 212-217.

Osgood, C. E. (1949). The similarity paradox in human learning: A resolution. Psychological Review, 56, 132-143.

Paige, J. M. & Simon, H. A. (1966). Cognitive processes in solving algebra word problems. In B. Kleinmuntz (Ed.) Problem solving: Research, method, and theory (pp. 51-119). New York: John Wiley and Sons.

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

Pea, R. D. & Kurland, D. M. (1984). Logo programming and the development of planning skills (Tech. Rep. No. 16). New York: Bank Street College of Education, Center for Children and Technology.

Petersen, C. G. & Howe, T. G. (1979). Predicting academic success in introduction to computers. AEDS Journal, 12, 182-191.

Polya, G. (1957). How to solve it. Princeton: Princeton University Press.

Polya, G. (1962). Mathematical discovery: On understanding, learning, and teaching problem solving (Vol. 1). New York: Wiley.

Polya, G. (1965). Mathematical discovery: On understanding, learning, and teaching problem solving (Vol. 2). New York: Wiley.

Quinn, J., Kirkman, J., & Schultz, C. J. (1983, September). Beyond computer literacy. Educational Leadership, 38-39, 67.

Reed, W. M. & Palumbo, D. B. (in press). The effect of the BASIC programming language on problem solving skills and computer anxiety. Computers in the Schools.

Reed, W. M., Palumbo, D. B., & Stolar, A. (in press). A comparison of the effects of BASIC and Logo instruction on problem solving skills. Computers in the Schools.

Reitman, J. S. (1976). Skilled perception in go: Deducing memory structures from inter-response times. Cognitive Psychology, 8, 336-356.

Resnick, L. B. & Ford, W. W. (1981). The psychology of mathematics for instruction. Hillsdale, NJ: Erlbaum.

Resnick, L. B. & Glaser, R. (1976). Problem solving and intelligence. In L. B. Resnick (Ed.) The nature of intelligence. Hillsdale, NJ: Erlbaum.

Sauter, V. L. (1986). Predicting computer programming skill. Computers and Education, 10 (2), 299-302.

Schoenfeld, A. H. (1982). Measures of problem-solving performance and of problem-solving instruction. Journal for Research in Mathematics Education, 13, 31-49.

Sheingold, K., Kane, J. H., & Endreweit, M. E. (1983). Microcomputer use in schools: Developing a research agenda. Harvard Educational Review, 53 (4), 164-172.

Shiel, B. A. (1981). The psychological study of programming. Computing Surveys, 13, 101-120.

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

Shneiderman, B. (1985). When children learn programming: Antecedents, concepts and outcomes. The Computing Teacher, 12 (5), 14-17.

Soloway, E., Lochhead, J., and Clement, J. (1982). Does computer programming enhance problem solving ability? Some positive evidence on algebra word problems. In R.J. Seidel, R.E. Anderson, and B. Hunter (Eds.) Computer literacy: Issues and directions for 1985. New York: Academic Press.

Suydam, M. (1980). Untangling clues from research on problem solving. In S. Krulick & R. E. Days (Eds.). Problem solving in school mathematics, (pp.34-50). Reston, VA: National Council of Teachers of Mathematics.

Szymezuk, M. & Frerichs, D. (1985). Using standardized tests to predict achievement in an introductory high school computer course. AEDS Journal, 18, 20-27.

Thorndike, E. L. (1924). Mental discipline in high school studies. Journal of Educational Psychology, 15, 1-22,83-98.

Thorndike, E. L. & Woodworth, R. S. (1901). The influence of improvement in one mental function upon the efficiency of other functions. Psychological Review, 8, 247-261, 384-395, 553-564.

Underhill, R. (1977). School mathematics for the 21st century: Evolving goals articulated as changing transfer expectations. In B.D. Felder (Ed.) Focus on the Future, (pp. 123-142). Houston, TX: University of Houston Press.

Webb, N. L. (1979). Processes, conceptual knowledge, and mathematical problem-solving ability. Journal for Research in Mathematics Education, 10, 83-93.

Webb, N.M. (1985). Cognitive requirements of learning computer programming in group and individual settings. AEDS Journal, 18, 183-193.

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

Wells, G. W. (1981). The relationship between the processes involved in problem solving and the processes involved in computer programming. Doctoral dissertation, University of Cincinnati, Cincinnati.

White, K. B. & Collins, R. W. (1983). An experimental investigation utilizing the computer as a tool for stimulating reasoning skills. AEDS Journal, 16, 234-243.

Wileman, S., Konvalina, J., Stephens, L. J. (1981). Factors influencing success in beginning computer science courses. Journal of Educational Research, 74 (4), 223-224.

Wolfle, L. M. & Ethington, C. A. (1985). GEMINI: Program for analysis of structural equations with standard errors of indirect effects. Behavior Research Methods, Instruments, & Computers, 17 (5), 581-584.

Zechmeister, E. B. & Nyberg, S. E. (1982). Human memory: An introduction to research and theory. Monterey, CA: Brooks/Cole.

APPENDIX A

Student Questionnaire

VIRGINIA TECH

COLLEGE OF EDUCATION
EDUCATION MICROCOMPUTER LAB

BLACKSBURG, VIRGINIA 24061
(703) 961-5587

February 16, 1987

Dear Student,

I am conducting a research study of mathematics problem solving and computer programming by high school students. To obtain information, I need to survey about 1000 students. I am asking each senior in your school (and several other schools) to answer the questions below and on the next two pages. This should only take about five minutes of your time.

All information will be kept strictly confidential. I will not use your name or the name of your school. If you have any questions about the study, please call me at the number above.

Your participation in this project is voluntary. You will not be penalized in any way if you choose not to answer the questions. Thank you for your help.

Sincerely,

Leah McCoy

* * * * * * * * * * * * * * * * *

I agree to participate in the research study described above. I give permission for my answers to the following questions and my SRA and DAT test scores to be used in the study. I understand that my name will not be used in any way.

_____
(Signed)

PLEASE PRINT:

1. Name_____

2. School_____

QUESTIONNAIRE - MATHEMATICS AND COMPUTER PROGRAMMING

DATE_____

1. Sex: [ ] Male [ ] Female

2. Do you have a computer at home? [ ] Yes [ ] No

3. List all mathematics courses you took in grades 9, 10, and 11. Include General Math, Consumer Math, Algebra, Geometry, or any other mathematics class.

Grade 9 _____

Grade 10 _____

Grade 11 _____

4. List all computer programming courses you have taken. Also give the school or camp where you took them and the year.

Class Title	Place	Year

5. How would you rate your computer programming skill?

Language	(MARK ONE FOR EACH LANGUAGE)			
	Strong	Average	Weak	Not at all
BASIC	[ ]	[ ]	[ ]	[ ]
Logo	[ ]	[ ]	[ ]	[ ]
Pascal	[ ]	[ ]	[ ]	[ ]
Other_____	[ ]	[ ]	[ ]	[ ]

6. Which of the categories below come closest to describing the jobs most recently held by your parents or guardians? (MARK ONE IN EACH COLUMN)

	Father, or Male Guardian	Mother, or Female Guardian
Do not live with father (or stepfather or male guardian)	[ ]	
Do not live with mother (or stepmother or female guardian)		[ ]
CLERICAL such as bank teller, secretary, typist, mail carrier, ticket agent	[ ]	[ ]
CRAFTSMAN such as baker, automobile mechanic, machinist, painter, plumber, telephone installer, carpenter	[ ]	[ ]
FARMER, FARM MANAGER	[ ]	[ ]
HOMEMAKER OR HOUSEWIFE ONLY	[ ]	[ ]
LABORER such as construction worker, car washer, sanitary worker, farm laborer	[ ]	[ ]
MANAGER, ADMINISTRATOR such as sales manager, office manager, school administrator, buyer, restaurant manager, government official	[ ]	[ ]
MILITARY such as career officer, enlisted man or woman in the Armed Forces	[ ]	[ ]
OPERATIVE such as meat cutter, assembler, machine operator, welder, taxicab, bus, or truck driver	[ ]	[ ]
PROFESSIONAL such as accountant, artist, registered nurse, engineer, librarian, writer, social worker, actor, actress, athelete, politician, but not including school teacher	[ ]	[ ]
PROFESSIONAL such as clergyman, dentist, physician, lawyer, scientist, college teacher	[ ]	[ ]
PROPRIETOR OR OWNER such as owner of a small business, contractor, restaurant owner	[ ]	[ ]
PROTECTIVE SERVICE such as detective, police officer or guard, sheriff, fire fighter	[ ]	[ ]
SALES such as salesperson, advertising or insurance agent, real estate broker	[ ]	[ ]
SCHOOL TEACHER such as elementary or secondary	[ ]	[ ]
SERVICE such as barber, beautician, practical nurse, private household worker, janitor, waiter	[ ]	[ ]
TECHNICAL such as draftsman, medical or dental technician, computer programmer	[ ]	[ ]
Never worked	[ ]	[ ]
Don't know	[ ]	[ ]

7. What was the highest level of education of your parents or guardians? (MARK ONE IN EACH COLUMN)

	Father, or Male Guardian	Mother, or Female Guardian
Do not live with father (or stepfather or male guardian)	[ ]	
Do not live with mother (or stepmother or female guardian)		[ ]
Less than high school graduation	[ ]	[ ]
High school graduation only	[ ]	[ ]
Vocational, trade, or business school after high school:		
Less than two years	[ ]	[ ]
Two years or more	[ ]	[ ]
College program:		
Less than two years of college	[ ]	[ ]
Two or more years of college (including two-year degree)	[ ]	[ ]
Finished college (four-or-five year degree)	[ ]	[ ]
Master's degree or equivalent	[ ]	[ ]
Ph.D., M.D., or other advanced professional degree	[ ]	[ ]
Don't know	[ ]	[ ]

APPENDIX B

Pearson Correlation Coefficients

Pearson Product-Moment Correlations for All Variables

	GENDER	SES	ABIL	HOMECO	MATH	PROG	MPS
GENDER	1.000	.026	-.084	.108	-.078	-.036	.099
SES	.026	1.000	.355	.289	.355	.141	.306
ABIL	-.084	.355	1.000	.158	.608	.358	.746
HOMECO	.108	.289	.158	1.000	.190	.192	.173
MATH	-.078	.355	.608	.190	1.000	.369	.555
PROG	-.036	.141	.358	.192	.369	1.000	.355
MPS	.099	.306	.746	.173	.555	.355	1.000

Notes: Pairwise deletion of missing SES data resulted in N's ranging from 593 to 800.
SES = Socioeconomic status,
ABIL = ability,
HOMECO = access to a home computer,
MATH = years of higher mathematics experience,
PROG = years of computer programming experience,
MPS = mathematics problem solving achievement.

The vita has been removed from the scanned document