The Effect of Problem-Solving Instruction on Computer Engineering Majors' Performance in Verilog (HDL) Programming

Hung, Yen-Chu

***Abstract*—**This study investigated the effect of instruction in problem-solving skills on computer engineering majors' performance in programming in the Verilog(HDL) language. Comparisons were made among two treatment groups (deduction and analogy) and a control group, whose pre- and post-test scores were analyzed with the ANCOVA procedure. Result showed that instruction in problem-solving skills significantly increased achievement in Verilog(HDL) language programming in computer engineering majors.

***Index Terms*—**Programming, analogy, deduction, Verilog (HDL) language, Problem solving

Computers are useful educational tools. Many schools have purchased microcomputers and many students are instructed in various programming languages. Research has shown that students are able to develop higher-order thinking skills through the process of programming [2],[36],[38]. For example, programming skills was found to improve learners' mathematics study skills. Through metacognitive monitoring, learners are able to correct small problems in their procedures (e.g., "debugging" or "stepwise refinement") while finding a solution to a programming problem. The purpose of the current study was to ascertain whether learning problem-solving skills (deduction and analogy) enables one to learn how to program in the Verilog(HDL) language. The result of this study could lead to a new way of looking at programming instruction.

Widespread individual studies have been conducted in schools on computer programming and cognitive development. For instance, Salomon and Perkins [38] conducted a study to ascertain the effect of LOGO language programming on cognitive development. Their study identified six kinds of transfer of learning from programming: (a) mathematical and geometric concepts and principles; (b) problem solving, problem finding, and problem management strategies; (c) abilities in formal reasoning and representation; (d) models of knowledge, thinking, and learning; (e) cognitive styles; and (f) enthusiasm and tolerance for meaningful academic engagement. Au and Leung [2] suggested that LOGO training has beneficial effects on children's higher level cognitive skills such as problem solving. Dalton and Goodrum [7] suggested that, when used together, computer programming and problem-solving strategy instruction may provide an effective means of teaching transferable problem-solving skills. Papert [30] noted that learning computer programming with LOGO is an ideal environment for learning problem-solving skills and increase the learner cognitive activity. These studies have shown that LOGO computer programming is an ideal environment for learning problem-solving skills because the LOGO language has (a) a top-down programming design; (b) modularity; and (c) requirements of limited use of logical constructs.

Research findings about other computer languages also confirm that programming portrays an ideal environment for learning problem-solving skills. Funkhouser and Dennis [10] indicated the effects of problem solving computer software on increasing problem-solving ability. Reed et al. [33],[34] found that learners using both the LOGO computer language and the BASIC computer language had significant increases in problem-solving skills. They also found no significant difference between using the LOGO language and the BASIC language in increasing problem-solving skills.

Several studies have been conducted regarding the implication of cognitive psychology related to designing programs. Hooper [14] reported that students using computer programming simulation employed more sophisticated algorithms during programming than did students who were not exposed to the manipulative computer model (MEMOPS) which was designed to facilitate the learning of programming. Thomas and Hooper [43] reported that simulation may be useful for reinforcing complex sequences. When using simulations the learner is forced to assume responsibility for executing the process, whereas in the alternative methods the learner responds to external questions or instructions. Alperson and O'Neil [1] compared a computer-based tutorial with simulations for transmitting knowledge in beginning anthropology and psychology courses. Salisbury [36] found that in the cognitive psychology area, the development of the use of subskills, inference, spaced practice, spaced review, the capacity of short-term memory, and the representation of information in memory are related to each issue in the design of computer drill programs.

II. A Basic Theoretical Model for Instruction

While the literature suggests that learning a computer programming language may improve a learner's problem-solving abilities, the reverse may also be true; that is, developing problem-solving skills may enhance the ability to learn a programming language. Both "problem solving" and "programming" involve a common subset of cognitive behaviors, memorizing and a schema or template. Thus, it may be inferred that each provides a set of experiences which enhance the learning of the other. Figure 1 depicts such a relationship. At the center of the figure are cognitive elements common to both programming and problem-solving proficiency. If a learner is deficient in one or more proficiency, he or she must acquire these structures. It has been suspected by some researchers that, for success in such activities, the learner must acquire these shared structures [1, 36, 43]. The mode for instruction in either activity may simply involve spending sufficient time and reinforcing the activities that build common cognitive structures.

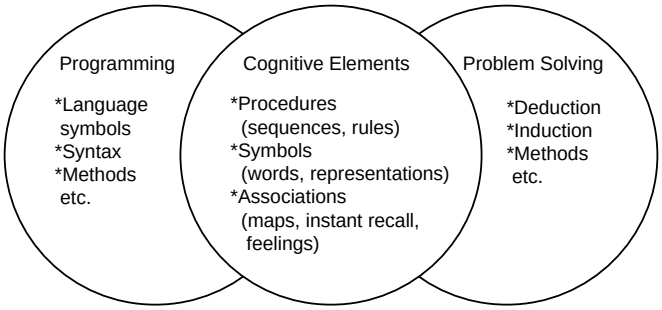

Fig. 1 Cognitive Elements Common to Problem Solving and Programming

The identification, description and validation of the common elements or structure are, by themselves, a major research problem. Support for this theory of "mutual causation" may be given by demonstrating that instruction in either problem solving or programming will enhance the acquisition of skill in the other. Some research results have indicated that learning problem-solving methods could increase student learning to program in the BASIC language. Palumbo and Reed [29] found a significant, positive correlation between problem-solving skills and BASIC language competency measures. Bayman and Mayer [4] reported learning BASIC programming involves the growth of syntactic and conceptual knowledge and that strategic knowledge and problem solving performance are strongly related to measures of conceptual knowledge.

Elements common to programming are language symbols, syntax, methods, etc., and those common to problem solving are deduction, analogy, analogy and methods, etc. All of the elements common to either programming or problem solving are also inherent as cognitive elements. For example, language symbols in programming are synonymous with the cognitive elements of symbols. Likewise, deduction in problem solving is a cognitive procedure, and so forth.

Purpose

To gain greater insight into the relationship between problem-solving and programming, the current study examined the effect of learning problem solving prior to learning to program in the Verilog(HDL) language. The purpose of this study was to compare the effects of learning problem-solving methods with instruction in a non problem-solving activity on subsequent achievement in learning to program in the Verilog(HDL) language in computer engineering students. The problem-solving methods studied in this research were deduction and analogy. This research may suggest that learning problem-solving methods could be important for students in providing them with increased knowledge and skills to learn how to program.

Research Questions

Seven research questions were developed to address the problem of the study. The first six sought to determine if there were significant differences:

1. among experimental and control groups on the Verilog(HDL) language program pretest mean scores;
2. between the problem-solving pretest mean scores of the experimental and control groups;
3. between the adjusted post-test means of the experimental and control groups on Verilog(HDL) language achievement;
4. between the adjusted post-test means of the experimental and control groups on problem-solving achievement;
5. between the adjusted post-test means of the experimental and control groups on the Verilog(HDL) language programming design; and
6. between the adjusted post-test means of the experimental and control groups on Verilog(HDL) language program understanding?

The last question sought to determine if there was a significant relationship:

7. between the Verilog(HDL) language tests and the problem-solving tests.

Problem Solving

Problem solving is a cognitive process of the brain at the higher cognitive layer that searches a solution or finds a path to reach a given goal [46],[48].Problem solving is one of the 37 fundamental cognitive processes modeled in the Layered Reference Model of the Brain[46]. This study was focused on deduction and analogy.

Deduction

Discrimination is the ability to determine the objects and/or events that have a direct impact on the problem. Students can go beyond the simple practice of discrimination and be led to solve classification problems and develop logical thinking skills in interesting ways. Skinner [39] noted that deduction is a way of constructing discriminative stimuli. Useful forms of deduction inspire the thinker to formulate a systematic form of analysis which will reduce the problems that exist to the simplest form. For example, a deductive inference could be stated as: A personal computer has a 1000 MHz processor. One can deduce that this computer is faster than computers with 333 MHz processors since 333 is less than 1000. Programming

often involves arranging statements into a logical sequence; for example,

```
Input  A,  B, C;
   Wire e;
   And g1(e, A, B);
```

is logical because before one can add and print the sum, the values must first be entered. The sequence

```
And g1(e, A, B);
   Input  A,  B, C;
Wire e;
```

would not be logical. Deductive reasoning would lead to this conclusion because of prior experiences in attempting to present a result without having the necessary information.

By arranging a problem into a series of logical steps, one applies "deductive" reasoning. Therefore, computer language programming requires extensive application of the principles of deduction. This computer programming strategy can be developed in several ways. Most appropriate to this research is to expand the students' ability to use deduction, which subsequently affects one's ability to design and understand computer language programming.

Analogy

Analogy is the comparison of two pairs which have the same relationship. An analogy is a comparison in which different items are compared point by point, usually with the idea of explaining something unknown by something known. Analogies are offered to provide insights, and can be very instructive. Analogies tend to suggest that existing similarities imply even more similarities. Analogy requires increased emphasis when considering its potential for creativity in solving new problems. Analogy thinking should expand one's considerations and remove barriers of fixed-rule thinking. Developing a knowledge base will improve the chance of finding a solution, and having the ability to draw on a broader knowledge base. Creative solutions are demanded of analogy thinkers. Such thinkers will have to organize, retrieve, and use an excess of information to solve their problems. Analogy uses experimental reasoning to arrive at the whole from the particulars. Thus, analogy employs basic inference strategy used in synthesized learning.

Creativity should not be construed to be limited to only analogy thinking. Creative efforts may involve both deductive and analogy thinking in the solution of problems of expression. Using problem-solving methods as a skill to understand and design computer language programming is the highest order commonly found in the demonstration of a designer's idea and expression of his or her aesthetic feelings. Thus, problem solving involves mostly intuitive, creative thought. A broad understanding of problem-solving capabilities and the logical combination of analogy as a skill along with one's personal knowledge broadens one's abilities to design and understand programming.

HDL(Hardware Description Language)

A hardware description language is a language that describes the hardware of digital systems in a textual form. It resembles a programming language, but is specifically oriented to describing hardware structures and behavior.

Effect of problem-solving methods on learning to program

Computer programming can be perceived as a problem solving process [20] that involves the following:

- Creating a program – Translating natural languages into Computer code to solve problems.
- Comprehending a program – Explaining computer code in natural languages.
- Modifying a program – Changing computer code to achieve a slightly different problem goal.
- Debugging a program – Fixing a non-working program.

The four computer programming activities in software engineer can be easily explained by the models and process presented in earlier sections. All the activities will involve relatively straightforward abstraction and conceptualization if the activities' tasks are problem with readily available memory cues for direct solution path determination.

To teach programming to students, research has shown a need for a more structured form of instruction to express the concept of program design. Linn [18] and Mayer [22] suggested that planning specific programs will effectively enhance learning computer language design. Plans can build program fragments that symbolize model action sequences in programming with particular tasks or subtasks. Researchers have used an expert model of programming to teach a beginning structured language such as PASCAL [17],[42]. Expert models of programming rely on plans as a central idea.

However, a mere theoretical plan for programming is not adequate. One must develop a reinforcing method for delivery that supports utilization of heuristics to improve inductive, deductive, metacognitive, and creative thinking as methods to apply when attempting problem solving. Programming schemata effectively requires the student to understand all the tools available and to use them in the most expeditious way to seek resolution. Vosniadou & Ortony [44] found that students who are exposed to analogs with surface similarity and deep similarity could induce a schema by the process of mapping. Then, the analogy schema (i.e., mapping identities) form the basis for analytical reasoning and problem solving. Based on this premise, problem solving can be taught and learned effectively. Problem solving helps students comprehend computer programming problems deeply which, in turn, helps them solve problems efficiently. Greeno and Simon [12] reported that patterns of information in a problem have to be recognized to determine that a problem-solving operation can be applied. McKeithen et al. [24] found that expert problem solvers represent problems immediately in terms of core programming structures which allow them to find elegant solutions. On the other hand, novice problem solvers fixate on surface features of problems without comprehending the structure which can be translated into a program. The current study deduces that programming skills are an effective

approach to teach problem-solving skills to improve students' ability to understand and solve critical thinking problems faced daily. In other words, students' problem representation and problem solving in teaching programming would be significantly improved by teaching schemata of programming.

Salomon & Globerson [37] reported that the degree of students' mindfulness as a tendency influences their learning. The more mindful a student is, the greater the capacity to learn computer programming. If students can be subsequently taught to use computer programming design after learning problem-solving methods, their ability to learn programming design will develop more rapidly. This study seeks to answer the following question: Is learning to design and understand Verilog(HDL) language programs more effectively achieved by those who first acquire problem-solving skills?

Does Learning Analogy and Deduction Affect Ability to Learn Programming?

Problem solving is commonly known as the application of acquired information, knowledge, and skills to new situation information acquisition itself is a cognitive process, which psychologists refer to as a transfer [23],[27],[31]. The transfer is in effect when what a person learned in one situation affects how the person learns and performs in a future situation.

Analogy and deduction are problem-solving methods. Analogy involves some set of cognitive processes that enables one to abstract rules from experience. Learning proceeds from the specific to a general rule. On the contrary, deduction is the set of processes used to apply rules that one has previously acquired. Learning involves the use of general knowledge to solve a specific problem. Thus, analogy is associated with the learning process, whereas deduction is associated with the application of knowledge. Pea [32] found that the best way to help students to learn computer programming is to provide clear models that show the process of controlling data. One could infer that students learn deduction (problem-solving skill) by using a step-by-step method to solve problems. Then, after learning computer programming, such as employing top-down model programming, one could use deductive methods or ideas to design and understand computer programming. Papert [30] and Feurzig et al. [9] reported that metacognition, general problem solving, and divergent thinking (analogy reasoning) have possible cognitive benefits toward active participation in computer programming by students.

Analogy reasoning ability influences programming achievement. Since students show differences in cognitive abilities (analogy reasoning ability) and mindfulness in computer learning, problem solving affects new task cognitive skills. These skills are directly related to design of programs. Thus, if one teaches students problem-solving methods using analogy or deduction, one could also increase their ability to understand and design computer language program. This was hypothesized in the present study.

Method

In this study an experimental research design was adopted in which 48 students enrolled in the Computer Engineering Department at National Chiayi University in Taiwan participated as subjects during the fall of 2004.

This research used a pretest/post-test control-group design structure [5]. Sections of students were randomly assigned to either one of two experimental groups, or to a control group. The method of instruction and testing is outlined in Table 1. Problem-solving and Binary Numbers were taught using traditional instructional methods. The primary instruction materials for the experimental groups (deduction and analogy) were developed from Mathematics for Elementary Teachers" by G. L. Musser and W. F. Burger [26] and submitted to a panel of experts for validation.

During the first week, all groups were given a Verilog(HDL) programming and problem-solving pretest prior to instruction. For the next two weeks, students in the deduction and analogy groups received common instruction in problem-solving while the control group learned Binary systems. During the fourth week, deductive and analogy reasoning was taught separately to students in the respective experimental groups while the control group continued learning Boolean Algebra and logic gates. A description of the instructional methods used for each group is presented in the following two subsections: Problem-solving instruction; and Instruction in deduction and analogy.

Problem-solving instruction

Two methods of traditional problem solving were taught to the experimental groups during the second to fourth weeks of instruction. The first method instructed students to understand the problem by asking seven questions:

1. Do you understand all the words;
2. Can you restate the problem in your own words;
3. Do you know what is given;
4. Do you know what the goal is;
5. Is there enough information;
6. Is there extraneous information; and
7. Is the problem similar to another problem you have solved?

Then students were taught to devise a plan by (a) looking for a pattern; (b) solving a simpler problem; and/or drawing a picture. Next, students were asked to carry out the plan by (a) implementing the chosen strategy (strategies) until the problem is solved or a new plan is made; (b) take a reasonable amount of time to solve the problem, seek hints, or put it aside; or (c) start over with a fresh strategy. Finally, students were instructed to look back and ask: (a) is the solution correct and does it satisfy the statement of the problem; (b) can an easier solution be found; and (c) can the solution be extended to a more general case? These traditional methods of problem-solving are based on the theory of George Polya, a mathematician who devoted his teaching to helping students become better problem solvers. An example of a problem given to students to solve using this approach is (provided below):

A teacher lineup contained four students (A, B, C, D), one of who is a gifted student. The lineup is graduated by

height, with the tallest student on the left and shortest on the right. There are two students between A and B, and C is left of D. The gifted student is third from the left, and B is to the right of the gifted student. Who is the gifted student?

A.. A
B B
C C
D D
E Cannot be determined

(Musser & Burger, 1988, p. 7)[26]

The second traditional method taught was to have students draw a picture, or diagram. A sample problem used was as follows:

A survey was taken of 150 college freshmen. Forty of them were majoring in mathematics, 30 of them were majoring in English, 20 were majoring in science, 7 had double majors of mathematics and English, and none had a double (triple) major with science. How many students had majors other than mathematics, English, or science?

Student were encouraged to discuss in groups and use Venn diagrams within a rectangle in solving this problem.

Another traditional problem-solving strategy taught to the students was direct reasoning. The following is a sample problem:

In a group of nine coins, eight weigh the same and the ninth is heavier. Assume that the coins are identical in appearance. Using a pan balance, what is the smallest number of weightings needed to identify the heavy coin?

Students discuss as a group their method of weighing the coins separated into three groups of three coins each. By direct reasoning, they weigh group A and B, and if they balance they determine the heavy coin is in group C. Reasoning that A and B are equal, they weigh B and C together and note which way the scale tips and arrive at the answer that two weightings are needed.

The third problem-solving strategy taught is to work backwards. A sample question used in this method is provided below:

How can you bring up from the river exactly six quarts of water when you have only two containers, a four-quart pail and a nine quart pail, to use for measuring?

Students discuss and visualize the given tools—the two containers. They imagine two cylindrical containers having equal bases whose heights are a 4:1 ratio. Then they eventually come to the realization that by filling the larger container to capacity (9 quarts), they can pour out exactly three quarts. Then they get the idea that they can achieve this by having just one quart in the smaller container. If they fill the larger container to full capacity twice and pour from it four quarts into the smaller container and the remainder into the river twice in succession, they can get one quart in the container. The answer was reached by using something already known and following the method of analysis, working backward.

Students were also taught to set up equations in mathematical symbols to do problem solving and then translate the language back to fit the real situation, thus using mathematical formulas (program design) and mathematical expression (computer language expression). They also learned to use analogy (similar objects), to decompose and recombine, and to use heuristics (the procedures of analysis and synthesis). In heuristics, one starts with analysis of what is required and taken for granted and draws conclusions from the consequences until a point is reached where synthesis can be used. Synthesis reverses the process, starting from that point last reached in analysis (what was admitted to be true) and what preceded it in the analysis, until the retraced steps lead to arriving at what was originally required. Synthesis is also called constructing a solution or reasoning.

Instruction in deduction and analogy

In deduction, premises are given and the problem solver must apply the appropriate rules to draw a conclusion. Deduction is a process of deriving a conclusion from one or more statements. A valid argument is an argument in which the conclusion must be true as long as the premises are true. In a deductive task, premises are given and the problem solver must apply the appropriate rules to draw a conclusion. For example, in an instructional session, the instructor might present the students the following two statements:

All students need to go to school.
Mary is a student.

Then the instructor would ask the students to discuss the statements and draw a logical conclusion: Mary needs to go to school (because she is a student). Generally, the students should not have much problem reaching this conclusion, but with harder problems, they may need direction from the instructor until they fully understand the process.

Analogy involves forming a general principle from the given facts or examples. In an analogy task, a series of instances are given and the problem solver must generate a rule or pattern that describes the structure of the problem. In an example lesson using analogy, the students are given several facts such as:

The day before yesterday, the sky was covered with dark clouds. It rained

Yesterday, the sky was covered with dark clouds. It rained. Today, the sky was covered with dark clouds. It rained.

Following discussion, the students are asked to tell the class whatever observations they made in regards to the phenomenon. The instructor may guide them to come forward with the principle that: when the sky is covered with dark clouds, it will rain.

Table 1. Instruction and Test Schedule

NOTE: Tests are shaded.

Week	Instruction Total by hours	Group		
		Deduction	Analogy	Control
1		*Verilog(HDL) programming and problem-solving pretest*		
	1	Problem-solving instruction	Problem-solving instruction	Word-processing
2-3	4	Problem-solving instruction	Problem-solving instruction	Word-processing
4	2	Deductive instruction	Analogy instruction	Word-processing
5-7	9	*Verilog(HDL)* programming instruction (everyone)		
8	1	*Verilog(HDL)* programming instruction (everyone)		
		Verilog(HDL) programming midterm test one		
9-11	9	*Verilog(HDL)* programming instruction (everyone)		
12	3	*Verilog(HDL)* programming instruction (everyone)		
		Verilog(HDL) programming midterm test two		
13-15	9	*Verilog(HDL)* programming instruction (everyone)		
16	3	*Verilog(HDL)* programming instruction (everyone)		
		Verilog(HDL) programming and problem-solving post-test		

knowledge pretests, two midterm tests, and two post-tests (Table 1). The teacher-made midterm tests were held after the teacher completed Verilog(HDL) language instruction. The midterm tests were composed of several problems from homework and textbook assignments. The final exams on problem solving and on Verilog(HDL) language programming covered all material taught in the Verilog(HDL) programming course.

Results

Comparisons were made among the three treatment groups. Approximately 10 percent of the pretest questions were answered correctly. As shown in Table 2 and Figure 2, the average pretest scores were 2.27, 2.14, and 2.40, respectively, for the control, deduction, and analogy groups. The students' knowledge of Verilog(HDL) language before instruction was limited; whereas following instruction approximately 30 percent of the post-test questions were answered correctly. The average post-test scores were 6.60, 9.42, and 11.13, respectively, for the control deduction, and analogy groups.

Table 2. Means for the Verilog(HDL) Language Pre- and Post-tests

	Means	
	Pretest	Posttest
Control	2.27	6.60
Deduction	2.14	9.42
Analogy	2.40	11.13

Comparisons were also made by treatments within groups (Table 2) In the pretest. Data from the tests were analyzed using the ANCOVA to determine if statistically significant differences existed among the groups (Table 3). The level of significance was set at $\alpha = 0.05$. The covariates were the Verilog(HDL) language pretest scores and the dependent variables were the Verilog(HDL) language post-test scores. There was a significant difference among the three treatments for the Verilog(HDL) language post-test, among the students in the three treatment groups.

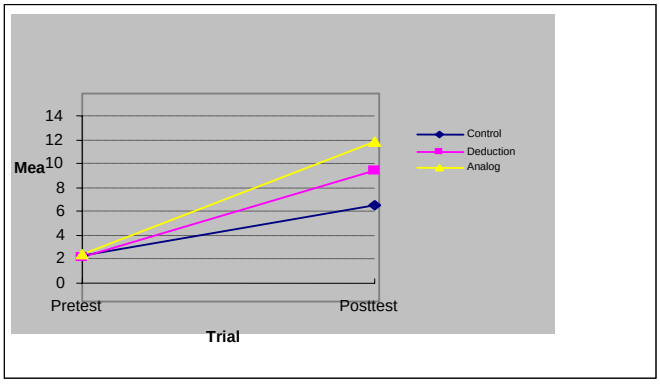

Figure 2. Pre- and Post-test Means for the Three Experimental Groups

Table 3 ANCOVA **for the Three Experimental Groups**

Source	df	Sum of Squares	Mean Squares	F value	Pr > F
Covariate	2	146.68	73.34	36.74	0.0001
Error	40	79.84	1.99		
Total	44	323.92			

Findings

The analysis of the pretest scores in Verilog(HDL) language concepts and problem solving showed that the randomly assigned groups were equal or nearly equal on these tests. There was a significant difference on the Verilog(HDL) post-test among the three treatment groups

The results indicated that when students first study problem-solving methods (analogy and deduction) they experience a significant increase in Verilog(HDL) language programming achievement (see Table 2 and Figure 2). The study also showed that students who first receive problem-solving instruction in analogy subsequently learn

Verilog(HDL) language programming significantly better than students who first receive problem-solving instruction in deduction and subsequently learn Verilog(HDL) language programming. Further evidence supports that male??? students in group one and two on Verilog(HDL) language programming in design and understanding performed significantly better than the female students in the control group.

Conclusions

This study investigated the effect of problem-solving instruction on computer engineering majors' performance in programming in the Verilog(HDL) language. The Verilog(HDL) language programming midterm and the Verilog(HDL) language programming post-test provided the means to assess achievement in Verilog(HDL) language program learning after two kinds of problem-solving instruction—analogy and deduction.

Evidence shows that students who first learn problem solving (deduction or analogy) followed by receiving instruction in Verilog(HDL) programming perform significantly better than students who use a non problem-solving method (word-processing) prior to learning the Verilog(HDL) language.

This study has implications for teaching programming and problem solving. It was theoretically proposed that there exists a mutual causation and interaction between problem solving and programming. This study provides support that learning a problem-solving method increases achievement in computer language programming. Other studies [2],[33],[[34], [38] support that learning computer language programming may improve a learner's problem-solving abilities. Combining these studies and the present study supports the mutual causation theory stemming from a common subset of cognitive behaviors, memories and schema or templates. Learning either problem-solving methods or programming provides a set of experiences which enhance the learning of the other.

References

[1] Alperson, J. R., & O'Neil, D. H. (1990, February). The boxscore: Tutorial 2, simulation 0. Academic Computing, 18-19, 47-49.

[2] Au, W. K., & Leung, J. P. (1991). Problem solving, instructional methods and LOGO programming. Journal of Educational Computing Research, 7(4), 455-467.

[3] B. Smith, "An approach to graphs of linear forms (Unpublished work style)," unpublished.

[4] Bayman, P., & Mayer, R. E. (1988). Using conceptual models to teach BASIC computer programming. Journal of Educational Psychology, 80(3), 291-298.

[5] Campbell, D. & Stanley, J. (1963). Experimental and quasi-experimental design for research. Boston: Houghton Mifflin.

[6] Chambers, J. A., & Sprecher, J. W. (1983). Computer-assisted instruction. Englewood Cliffs, NJ: Prentice-Hall, Inc.

[7] Dalton, D. W., & Goodrum, D. A. (1991). The effects of computer programming on problem-solving skills and attitudes. Journal of Educational Computing Research, 7(4), 483-506.

[8] E. H. Miller, "A note on reflector arrays (Periodical style—Accepted for publication)," *IEEE Trans. Antennas Propagat.*, to be published.

[9] Feurzig, W., Horowitz, P. & Nickerson, R. (1981). Microcomputers in education. Cambridge, MA: Bolt, Beranek, and Newman.

[10] Funkhouser, C., & Dennis, J. R. (1992). The effects of problem-solving software on problem-solving ability. Research on Computing in Education, 24(3), 339-348.

[11] G. O. Young, "Synthetic structure of industrial plastics (Book style with paper title and editor)," in *Plastics*, 2nd ed. vol. 3, J. Peters, Ed. New York: McGraw-Hill, 1964, pp. 15–64.

[12] Greeno, J. G., & Simon, H. A. (1988). Problem solving and reasoning. In R. C. Atkinson, R. J. Herrnstein, G. Lindzey, & R. D. Luce (Eds.), Stevens' Handbook of Experimental Psychology: Learning and cognition (2nd ed.).. New York: John Wiley & Sons.

[13] H. Poor, *An Introduction to Signal Detection and Estimation*. New York: Springer-Verlag, 1985, ch. 4.

[14] Hooper, E. J. (1986). Using programming protocols to investigate the effects of manipulative computer models on student learning. (Doctoral dissertation, Iowa State University, 1986). Dissertation Abstracts International, 47, 3009A.

[15] Kahney, H. (1993). Problem solving: Current issues. Philadelphia, PA: Open University Press.

[16] Langstaff, J. J. (1989). Problem representation and achievement in computer programming: The differential effects of inductive reasoning skills and computer programming experience. (Doctoral dissertation, University of Iowa). Iowa City, IA.

[17] Linn, M. C. & Clancy, M. J. (1989). The case for case studies of programming problems. Paper presented at the meeting of the American Education Research Association, San Francisco, CA.

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

[19] Luchins A. S. & Luchins, E. H. (1970). Wertheimer's seminars revisited problem solving and thinking. Albany, NY: State University of New York at Albany, Inc.

[20] Matlin, Margaret W. (1998), Cognition, Fourth Edition. Harcourt Brace & Company, ISBN:0-15-504081-2, pp.354-372.

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

[22] Mayer, R. E. (1988). Introduction to research on teaching and learning computer programming. In R. E. Mayer(Ed.) Teaching and learning computer programming. (pp. 1-12) Hillsdale, NJ: Lawrence Erlbaum Associates.

[23] Mayer, Richard E. (1992), Thinking, Problem Solving, Cognition, Second Edition, W.H. Freeman and Company, ISBN:0-7167-2215-1, pp. 36-38, 167-202, 397-400.

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

[25] Michalski, R. S., (1983). Theory and methodology of inductive learning. In R. S. Michalski, Machine learning: An artificial intelligence Tioga Publishing Co.

[26] Musser, G. L., & Burger, W. F. (1988). Mathematics for elementary teachers. New York: Macmillan.

[27] Ormrod, Jeanne Ellis (1999), Human Learning, Third Edition, Prentice-Hall, Inc, Simon & Schuter/A Viacom Company, ISBN: 0-13-875684-8, pp. 347-383.

[28] Palumbo, D. B., & Reed, W. M. (1991). The effects of BASIC programming language instruction on high school students' problem -solving ability and computer anxiety. Journal of Research on Computing in Education, 23(3), 343-372.

[29] Palumbo, D. B., & Reed, W. M. (1992). The effects of BASIC instruction on problem solving skills over and extended period of time. Journal of Research on Computing in Education, 8(3),311-325

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

[31] Payne, David G., Michael J. Wenger (1998), Cognitive Psychology, Houghton Mifflin Company, Boston, New York, ISBN: 0-395-68573-7, pp 455, 479-482.

[32] Pea, R. D. (1986). Language-independent conceptual 'bug' in novice programming. Journal of Educational Computing Research, 2(1), 25-35.

[33] Reed, W. M., & Palumbo, D. B. (1987/1988). The effect of the BASIC programming language on problem-solving skills and computer anxiety. Computers in Schools, 4(3/4) 91-104.

[34] Reed, W. M., Palumbo, D. B., & Stolar, A. L. (1987/1988). The comparative effects of BASIC and Logo instruction on problem-solving skills. Computers in Schools, 4(3/4), 105-118.

[35] Ricardo, C. M. (1983). Identifying student entering characteristics desirable for a first course in computer programming. Doctoral dissertation. Columbia University, NY.
[36] Salisbury, D, F. (1990). Cognitive psychology and it implications for designing drill and practice programs for computers. Journal of Computer-Based Instruction, 17, 23-30.
[37] Salomon, G, & Globerson, T. (1987). Skill may not be enough: The role of mindfulness in learning and transfer. International Journal of Educational Research, 7, 623-637.
[38] Salomon, G., & Perkins, D. (1985). Transfer of cognitive skills from programming: When and how? Journal of Educational Computing Research, 3(2), 149-169.
[39] Skinner, B. F. (1968). An operant analysis of problem solving. In B. Klemmuntz (Ed.), Problem solving: Research, method and theory. New York: Wiley.
[40] Skinner, B. F. (1968). The technology of teaching. New York: Appleton Century Crofts.
[41] Snow, R. E. (1980). Aptitude Processes. In R. E. Snow, P., A., Federico, & W. E. Montague (Eds.) Aptitude learning and instruction: Cognitive process analyses of aptitude (Vol.1). Hillsdale, NJ: Erlbaum.
[42] Soloway, E. (1986). Learning to program = learning to construct mechanisms and explanations. Communications of the ACM, 29, 850-858.
[43] Thomas, R. A., & Hooper, E. (1991). Simulation: An opportunity we are missing. Journal of Research on Computing in Education, 23(4), 497-513.
[44] Vosniadou, S., & Ortony, A. (1989). (Eds.). Similarity and analogical reasoning. New York: Cambridge University Press.
[45] W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.
[46] Wang, Yingxu, Ying Wang, S Patel, and D. Patel (2004), A layered Reference Model of the Brain, IEEE Transactions on Systems, Man, and Cybernetics, Vol. 34, to appear.
[47] Wertheimer, M. (1945). Productive thinking. New York: Harper & Row.
[48] Wilson, R.A. and F. C. Keil (2001), The MIT Encyclopedia of the Cognitive Science. MIT Press.

Yen-Chu Hung is an Associate Professor of Computer Engineering and the Director of the Computer Center at National Chiayi University in Taiwan. He specializes in computer languages and instruction.