---Invited Paper---

To appear in the proceedings of the
Conference on National Computer Literacy Goals
for 1985

Does Computer Programming Enhance Problem Solving Ability? Some Positive Evidence on Algebra Word Problems

Elliot Soloway **
Jack Lochhead *
John Clement *

* Cognitive Development Project
Department of Physics and Astronomy
University of Massachusetts
Amherst, Mass., 01003

** Department of Computer and Information Science
Unversity of Massachusetts
Amherst, Mass., 01003

* This research was supported in part by NSF Grant SED78-22043 in the Joint National Institute of Education - National Science Foundation Program of Research on Cognitive Processes and the Structure of Knowledge in Science and Mathematics.

** This research was also supported in part by a grant from the U.S. Army Research Institute for the Behavioral and Social Sciences, ARI Grant No. DAHC-19-77-G-0012.

This paper reviews certain research conducted at the Univ. of Mass. over the past 2 years. It includes material reported in earlier reports, particularly, [Clement, Lochhead and Soloway 1979, 1980].

ABSTRACT

There is a common intuition among those in computer science that programming helps to develop good problem solving skills. In this paper we review our attempts to isolate the specific factors in programming which enhance mathematical problem solving ability. In previous studies we found that a surprising number of college students have difficulty with very simple algebra word problems. We have also shown that significantly more students are able to solve these word problems correctly in the context of writing computer programs, than in the context of simply writing an algebraic equation. We obtained similar results in comparing the reading of algebraic equations within computer programs and the reading of algebraic equations by themselves. These observations suggest that computer programming may focus students' attention on the active, procedural semantics of equations, a view that many students fail to take naturally.

We have also conducted video-taped interviews with students as they solved typical algebra word problems. Analysis of this data supports the hypothesis stated above. Based on this data, a competing hypothesis has also been discounted; namely, it is not the case that students do better in the programming context simply because they check their answers more carefully.

We conclude with speculation on the impact of our results for education. We suggest that problem solving literacy can be promoted by teaching from an integrated algebra/programming curricula. Finally, based on companion research, we go on to suggest some cognitive factors important to the choice of which programming language should be taught, and how it should be taught.

I. Introduction

There is a common intuition among those in computer science education that programming encourages the development of good problem solving skills. Papert [1971] and the LOGO project were early proponents of this view; they developed a method to teach geometry by way of computer programming. Underlying their view is a recognition of the importance of "doing," of activity, and of procedure. Educators from Dewey to Piaget have emphasized that in order to understand a concept students need to take an active role.

This pedagogical intuition needs to be investigated empirically so that it can be articulated more precisely. A step in that direction was made recently by Howe, O'Shea, and Plane [1979] in a series of experiments based on the following paradigm: a course in mathematics was taught in the standard way without incorporating computer programming, and simultaneously, the same course was taught with computer programming. Students' mastery of the subject matter was then compared across the two groups. In such experiments Howe et al. obtained a consistent effect in favor of computer programming.

The above work might be characterized as experiments on the "macro" level; in contrast, the work reported here has focussed on the "micro" level. That is, we have attempted to develop <u>tools</u> which would enable us to isolate specific, critical factors contributing to the above results. Rather than studying an entire course, we have focussed on single problems. We shall present results (section II) concerning the surprisingly poor performance of college students on two ostensibly simple algebra word problems. These results suggest

several hypotheses. One is that the errors resulted from the students' failure to give a procedural interpretation to the algebraic equation. A second set of experimental results (section IV) provides significant, new support for this hypothesis. Namely, students do signficantly better on certain algebra word problems when they occur in a programming context, than when the same problems occur in a traditional, algebraic (non-programming) context. Based on analyses of group test data, and on video-taped student interview data, we suggest several factors involved in programming which could account for the way in which this activity fosters a more active interpretation of algebra by students. We conclude with a discussion of the implications of this work for education.

II. Experiments with Word Problems in a Traditional Algebraic Setting

In a previous study, Clement, Lochhead, and Monk [1980] uncovered several types of ostensibly simple problems which gave students great difficulty. In Table 1 we list two of these problems and typical performance results. Among engineering students 37% missed the first problem while 73% missed the second! In their experiments, Clement, Lochhead and Monk (1980) were able to eliminate algebraic manipulation and "tricky wording" as sources for the errors.

The errors made on problems 1 and 2 were largely of one kind; most were "reversals": $6S = P$ instead of $S = 6P$ and $4C = 5S$ instead of $5C = 4S$. The consistency of

these error patterns argues against the idea that they were caused simply by carelessness, and suggests that they stem from a common conceptual bug.

III. Interpretation of Algebra Experiments

How is it possible for students with such weaknesses to survive high school and college science courses? It appears that these students have developed special purpose translation algorithms which work for many textbook problems, but which do not involve anything that could reasonably be called a semantic understanding of algebra. Many word problems are constructed so that they can be solved through a trivial word-to-symbol matching algorithm. Others, such as physics problems, are given in a highly restricted context, where there are only two or three pretaught equations to choose between. This choice can be made either by picking the one equation which contains all of the given variables or through units analysis. While these techniques may be partially successful in many classroom situations, they are too primitive and unreliable to be trusted in any but the most routine applications.

In order to pursue the source of these errors, we conducted audio and video-taped interviews with 20 students who were asked to think out loud as they worked these and other related problems. On the "Students and Professors " problem we were able to identify two strategies which led to the reversal error. In the first, the student simply assumed that the order or contiguity of key words in

the English language problem statement mapped directly into the order of symbols appearing in the algebraic equation. Weaknesses in this type of direct translation strategy have previously been analyzed by Paige and Simon[1966].

On the other hand, in a second incorrect strategy, students acted as if they did use an accurate representation of the meaning of the problem. However, reversal errors appeared to arise because of confusion about the semantics of the algebraic equation. For example, one subject wrote '6S = 1P' and explained:

> "There's six times as many students, which means it's six students to one professor and this (points to 6S) is six times as many students as there are professors (points to 1P)."

When asked to draw a picture to illustrate his equation, the student drew from right to left one circle with a 'P' in it, an equal sign, and six circles with "S's" in them. Subjects such as the above seem to use an accurate model of the practical situation, but they still fail to symbolize that understanding with the correct equation.

Apparently such subjects interpret the reversed equation, '6S = P', as stating that a large group of students are associated with a small group of professors. To these students the letter "P" stands for "a professor" rather than "the number of professors" and the <u>equal sign expresses</u> a <u>comparison</u> or <u>association rather than</u> an <u>equivalence</u>. The fact that the "S" side of the equation has a "6" on it indicates that it is larger than the "P" side which has no modifier. Thus, there appear to be more S's than there are P's. Thus the student attempts to write the algebraic equation '6S = P' as a "figurative" statement, describing a <u>passive picture</u> in which

relative sizes of the entities are represented.

This contrasts to the correct equation 'S = 6P', which needs to be viewed as expressing an active operation being performed on one number (the number of professors) in order to obtain another number (the number of students). The correct equation, S = 6P, does not describe sizes of the groups in a literal or direct manner. Rather, it describes an equivalence relation that would occur if one were to make the group of professors six times larger. In other words, the equation S = 6P is not a direct description of the actual situtation, but rather, it represents the hypothetical state of affairs which would result after performing the operation of multiplying the current number of professors by 6. The key to fully understanding the correct translation lies in viewing the number six as an operator which transforms the number of professors into the number of students. For example, one subject who correctly wrote S = 6P said:

> "If you want to even out the number of students to the number of professors, you'd have to have six times as many professors."

The equation is thus interpreted in a procedural manner as an instruction to act.

IV. Computer Programs vs. Algebraic Equations: Experimental Results

On the basis of the foregoing analysis, we developed the following hypothesis: if students were placed in an environment

which could induce them to take a more active, procedural view of equations, then the error rate on these problems should go down. One clear candidate for such an environment is that of computer programming. That is, a computer program is a definite prescription for action; it is a set of commands which produces some result. Below, we present empirical tests of this hypothesis; in the next section we shall present our analysis of these results.

Experiment 1

In this experiment, our subjects were primarily freshmen and sophomores in a course on machine and assembly language programming. Half the class was given problem 1 in Table 2, while the other half was simultaneously given problem 2 in Table 2. The only difference in the questions is that the latter asks for a computer program while the former asks for an algebraic equation. As indicated in Table 2, significantly more students could solve problem 1 than could solve problem 2. Probability of these results on the assumption that errors on each problem were equally likely is $p < .05$.

Experiment 2

The above experiment explored the writing of computer programs or equations. However, Clement, Lochhead and Monk [1980] observed that reading equations also gave students a great deal of trouble. That is, many students failed to write a correct explanation of the

relationship expressed by the equation. Following the hypothesis outlined above, we wanted to compare the results of students reading and explaining an equation, which was embedded in a computer program, with students reading and explaining an equation, which stood alone. The two questions in Table 3 were given as part of an 11 question test to 87 freshman, engineering students. The difference between the groups which answered one correctly but the other incorrectly is quite interesting. Namely, the group of students who answered the computer problem correctly (problem 2, Table 3), but the equation problem incorrectly (problem 1, Table 3) was more than 3 times as large as the group who answered the equation problem correctly, but missed the computer problem. This difference is significant at the .005 level. Here again, we see that the programming environment facilitated the students' understanding.

V. Why a Programming Context Decreases Reversal Errors: Some Hypotheses

Based on the experiments described earlier, we developed a set of hypotheses (see Table 4) that could explain the performance difference discussed above. Other researchers have commented that they felt that hypothesis 4 was the key reason for the performance difference. That is, they felt that "checking one's answer by substituting numbers" was the factor that enabled students to write the correct equation. This technique —— writing a program down, and "running it" with test data for verification —— is emphasized in programming courses. Moreover, if the "checking" hypothesis were

correct, then the contribution of the programming environment per se would be minimal; one could "run" straight algebraic equations, though this technique is not stressed in instruction.

In order to better understand the factors involved, we employed the same technique that had provided valuable insights in the algebra study (Clement, Lochhead, and Monk [1980]), namely, video-taping students as they solved problems. The same written test discussed in Experiment 2 above was given to students who were taking a second course in computer programming. We selected for video-taping, 5 students who missed the algebra version of the problem on the written test. During the video-taping session, we asked each student to solve 4 problems, of the type discussed earlier. Half these students were given the problems in the following order: an algebra version, a programming version, an algebra version, and a programming version. The other half started alternating with the programming version. Of the 5 students interviewed, 3 "flip-flopped" on their answers at least once, i.e., they got the algebra wrong and the programming correct (or the programming correct and the algebra wrong), and some did this more than once. In some cases, the opposing solutions appeared within 30 seconds to 1 minute of each other, with the student's written work appearing on the same piece of paper. It is interesting to note, though, that all 3 students did answer the last (fourth) problem correctly.

In the video-taped interviews with these students, we did not see any indication that number checking/substitution was the main factor contributing to a correct program solution. It appeared that

other factors in the programming environment were more important. For example, the stuents regularly and systematically put a READ statement at the top of the program. This appeared to "trigger" the concept of input, which in turn, triggered the concept of variable, i.e., a place that holds a value, a number. Thus they were less likely to fall into the misconception of thinking of S as a label for "a student." [1] As they wrote the program, or after it, the students repeatedly gave "qualitative" explanations for their correct answer. For example, one subject said:

> Slightly more bought hamburgers than bought sandwiches... So I knew 6/7 wouldn't quite work so it must have been 7/6...
> I knew the ratio of people getting hamburger to sandwich is just a little bit more than one so I was looking for a fraction using the numbers in the problem which would be slightly more than 1.

After the correct equation was written down, we did see some students checking the equation with numbers; however, this came after the students appeared quite confident with their equation, as evidenced by explanations such as the above.

At this point our analyses and conclusions (e.g., the five hypotheses in Table 4) are in flux. We are, for example, investigating the hypothesis that students have different "spheres" or "frames" of knowledge for algebra and programming. When asked to solve a problem using a program, the student may be switching to a new knowledge base, somewhat independent of the algebra knowledge

---

[1] By "labels" we mean that students appeared to treat "S" and "P" as they would "feet" and "yards" in the following relationship "3 feet = 1 yard."

base. We are currently analyzing more video-tape data in order to evaluate and refine our hypotheses.

VI. Implications for Education

The data of Clement, Lochhead, and Monk [1980] show that it is particularly important to distinguish between the math courses a student has taken and the knowledge of mathematics the student actually possesses. If one defines mathematical literacy as the ability to translate back and forth between problems in the world and mathematical descriptions, then there is good reason to question whether mathematics instruction has promoted such literacy.

This is not merely an academic question. Recent surveys suggest that Russian college students are at least two years ahead of their American counterparts in the level of their mathematics coursework. Like the missile crisis of the 1960's, this information could spur a major national effort to catch up. But if Russian math courses are no more effective in teaching students how to think mathematically than are our own, we may only be competing to see who can get further behind. We do not mean to belittle the potential threat of a math-gap. On the contrary we see it as a very serious issue which must be studied seriously and quickly.

Based on our work, we can make two proposals which might facilitate "problem-solving literacy." The first is obvious: integrate a course(s) on programming into the mathematics curriculum. The assumption underlying this recommendation is that "transfer" will

occur -- the skills learned in the programming course will resurface in the mathematics courses. While our data does not conclusively show transfer, we have results which are consistent with that hypothesis. In particular, recall that all students who were interviewed on video-tape did eventually solve the test problems correctly. Whether this was a practice effect or a true transfer is unclear. We are currently planning a series of experiments to further explore the "transfer" question.

Our second proposal is more radical: redefine much of the early "mathematics" curricula to be programming based. That is, teach algebra as an integral part of programming. Clearly, the students would learn the same symbol manipulation skills that they develop in the algebra course, but in addition, they would be in a better position to develop problem-solving literacy, which is the goal of the former courses. Transfer would be less of an issue, since algebra and programming would be integrated at the start! We believe that this approach would help students develop a much deeper understanding of the concepts of variable and function, and a much greater facility with mathematical modelling in practical situations. Clearly, this suggestion is not new; it has been put forth, in various forms, by Papert [1979].

VI.1 A Cognitively Appropriate Programming Language

If programming should be included in the curricula, the question of "which language" to use comes up. Here again, the style of research employed in the above studies could play a key role. That

is, with the precious little empirical research into the cognitive factors involved in programming [Seidel and Hunter 1970, Mayer 1979, Miller 1976, Ledgard 1979], and with the current push in the direction of more "formality" in programming, programming languages are in danger of becoming like mathematics, i.e., developing a notation system that students can learn, but only with difficulty, and can manipulate, but without any real understanding. We have begun an empirical study, using the group testing and individual video-taping techniques described above, to explore how students learn to program in Pascal, a programming language which has recently gained wide acceptance.

Our preliminary results are quite unsettling (Soloway, et al. [1980]). We conducted a pilot study of students in a summer school course on introductory Pascal programming. During the course, we collected "on-line protocols" of students as they interacted with the Pascal system in order to solve homework assisgnments, and we administered a group test at the end of the school semester. Based on an analysis of this data, we found, for example, that students were confused in many ways about the basic iteration constructs in Pascal. They did not seem to understand -- or trust -- the complex functions which the for, while, and repeat constructs perform. Students either made explicit those functions which are implicitly performed (e.g., loop end testing, index variable incrementing) or they assumed that more actions would be performed than actually are (e.g., incrementing the index variable in the while construct). Since we believe that one of the primary benefits of programming is

that it requires students to make each step explicit, we are suspicious of the blind initial use of sophisticated constructs which have many implicit operations. We speculate that in fact students do not have mental models of the primitives which compose these higher level constructs. This suggests that they should be taught iteration first using the primitives, and then graduate to the complex constructions, as was often done in teaching BASIC and FORTRAN.

VII. Concluding Remarks

We have attempted to empirically explore the contribution of programming to problem solving in the context of ostensibly simple word problems. Our general hypothesis was that programming does enhance problem solving, because programming encourages the needed procedural view. We have carried out a number of studies on this question (Clement, Lochhead, and Soloway [1980]), many of which are reported here. To date, all support our initial hypothesis. Clearly, however, much more needs to be done. In particular, our work has focussed on a limited context, and issues such as transfer must be further explored. Nonetheless, we are encouraged by the results of these studies, and are planning further experiments in order to broaden and deepen our hypotheses. Finally, based on this research, we have put forth two suggestions which call for the "wedding" of mathematics and programming. While many questions remain to be researched, we have before us the exciting prospect of uncovering new modes of instruction many times more powerful than those we now employ.

References

Clement, J., Lochhead, J., and Monk, G. (1980) "Translation Difficulties in Learning Mathematics," American Mathematical Monthly, in press.

Clement, J., Lochhead, J. and Soloway, E. (1979) "Translating Between Symbol Systems: Isolating Common Difficulty in Solving Algebra Word Problems," COINS Technical Report 79-19, Department of Computer and Information Science, University of Massachusetts, Amherst.

Clement, J., Lochhead, J. and Soloway, E. (1980) "Positive Effects of Computer Programming on Students' Understanding of Variables and Equations," Proc. of the National ACM Conference, Nashville.

Howe, J.A.M., O'Shea, T. and Plane, J. (1979) "Teaching Mathematics Through Logo Programming," DAI Research Paper 115, Department of Artificial Intelligence, University of Edinburgh.

Kaput, J. (1979a) Personal communication.

Kaput, J. (1979b) "Mathematics and Learning: Roots of Epistemological Status," Cognitive Process Instruction (J. Clement and J. Lochhead, Eds.), Franklin Institute Press, Philadelphia.

Ledgard, H., Whiteside, J., Singer, A. and Seymour, W. (1979) "Report on an Experiment on the Design of Interactive Command Languages," COINS Technical Report 79-21, Department of Computer and Information Science, University of Massachusetts, Amherst.

Mayer, R.E. (1979) "A Psychology of Learning BASIC," Comm. of the ACM, November.

Monk, G.S. (1979) Personal communication.

Paige, J. and Simon, H. (1966) "Cognitive Processes in Solving Algebra Word Problems," Problem Solving Research, Method and Theory (B. Kleinmutz, Ed.), John Wiley and Sons, New York.

Papert, S. (1971) "Teaching Children to be Mathematicians versus Teaching about Mathematics," MIT AI Lab Memo 249, Cambridge.

Papert, S. (1979) "Computers and Learning," in The Computer Age: A Twenty-Year View, (M. Dertouzos and J. Moses, Eds.), The MIT Press, Cambridge, Mass.

Seidel, R.J. and Hunter, H.G. (1970) "The Application of Theoretical Factors in Teaching Problem-Solving by Programmed Instruction," International Review of Applied Psychology, Vol. 19, No. 1, April.

Soloway, E., Bonar, J., Barth, J. and Woolf, B. (1980) "Conceptions and Misconceptions in Students' Understanding of Basic Programming Issues," in preparation.

Problem 1:

Write an equation using the variables S and P to represent the following statement: "There are six times as many students as professors at this University." Use S for the number of students and P for the number of professors.

Sample Size	% Correct	% Incorrect
150	63	37

Problem 2:

Write an equation using the variables C and S to represent the following statement: "At Mindy's restaurant, for every four people who order cheesecake, there are five people who ordered strudel." Let C represent the number of cheesecakes and S represent the number of strudels.

Sample Size	% Correct	% Incorrect
150	27	73

Problem 3:

Spies fly over the Norun Airplane Manufacturers and return with an aerial photograph of the new planes in the yard.

They are fairly certain that they have photographed a representative sample of one week's production. Write an equation using the letters R and B that describes the relationship between the number of red airplanes and the number of blue planes produced. The equation should allow you to calculate the number of blue planes produced in a month if you know the number of red planes produced in a month.

Sample Size	% Correct	% Incorrect
34	32	68

Table 1

Problem 1:

Given the following statement:

"At the last company cocktail party, for every 6 people who drank hard liquour, there were 11 people who drank beer."

Write a computer program in BASIC which will output the number of beer drinkers when supplied (via user input at the terminal) with the number of hard liquour drinkers. Use H for the number of people who drank hard liquour, and B for the number of people who drank beer.

Sample Size	% Correct	% Incorrect
52	69	31

Problem 2:

Given the following statement:

"At the last company cocktail party, for every 6 people who drank hard liquour, there were 11 people who drank beer."

Write an equation which represents the above statement. Use H for the number of people who drank hard liquour, and B for the number of people who drank beer.

Sample Size	% Correct	% Incorrect
51	45	55

Probability of these results on the assumption that errors on each problem were equally likely is $p < .05$

Table 2

Problem 1:

Write a sentence in English that gives the same information as the following equation:

$$A = 7S$$

A is the number of assemblers in a factory.
S is the number of solderers in a factory.

Problem 2:

```
Program Kayak
Input I
K = I * 2
Print K
End
```

For the above computer program describe in English the mathematical relationship which exists between I, the number of Igloos, and K, the number of Kayaks.

Comparison of Problem 1 and Problem 2

a.	Number of people who got 1 correct, but 2 incorrect	5
b.	Number of people who got 2 correct, but 1 incorrect	18

Probability of these results on the assumption that case a and b were equally likely is < .005

Table 3

1. Unambiguous semantics of programming language constructions. While various mathematical symbols (e.g., the equals-sign) are often open to a variety of interpretations in mathematics (see [Kaput 1979b]), programming languages require that only one interpretation be associated with each symbol. This fact is usually emphasized in programming language instruction. For example, the meaning of '=' in 'I = I + 1' is explicitly defined as an act of replacement, i.e., the value of the right side of the equation becomes the new value of the variable on the left. Also, the interpretation of variables is clear, i.e., they stand for numbers which are acted on by operators.

2. Explicitness required by the suntax of programming languages. The fact that one must write '6*S' rather than simply '6S' might serve to prompt one to view that expression operatively as meaning "six times the number of students" rather than falling into the error of viewing it descriptively as "six students".

3. Viewing an "equation" in a programming language as an active input/output transformation. That is, the right hand side of the equation (the input) is operated on to produce a value for the left hand side (the output).

4. The practice of debugging programs. While students may not be encouraged to "run their equations" in typical mathematics courses, this concept of actual number testing is an integral part of programming and programming education.

5. The practice of decomposing a problem into explicit steps. A number of students solved the computer program problem by writing down a two step sequence of operations,

```
X = B/6
B = 11*X
```

One interpretation for this phemenon might be that students "saw" partial results "produced" on the way to the solution.

Table 4

Preliminary Hypothesis: Why Programming Facilitates Problem Solving