Scaffolding Effective Problem Solving Strategies in Interactive Learning Environments

Douglas C. Merrill
The RAND Corporation
1700 Main Street
Santa Monica, CA 90407-2138
Doug_Merrill@rand.org

Brian J. Reiser
Institute for the Learning Sciences and
School of Education and Social Policy
Northwestern University
2115 N. Campus Drive
Evanston IL 60208-2610
reiser@ils.nwu.edu

Abstract

Novices often experience great difficulty learning new domains. Thus, understanding how best to scaffold novice problem solving has potentially tremendous importance for learning in formal domains. In this paper, we present results from an experimental study that compared learning outcomes of students solving introductory programming problems in three different learning environments. This range of environments varies in two ways. First, the notations used in the environments vary between diagrammatic and textual. More importantly, the environments differ in the cognitive activities students are led to perform while solving problems, such as prediction of intermediate results and noting future goals to achieve. This experiment demonstrated that environments that scaffold more of the important cognitive activities lead to superior performance, regardless of whether the environments are textual or diagrammatic.

Introduction

Although learning by doing is generally viewed as superior to more passive learning situations, learning by solving problems can also give novices significant difficulties. This has led a great deal of cognitive science research to focus on methods for scaffolding novices in their attempts to master new knowledge. To create effective learning environments, designers must focus upon the strategies novices use during problem solving. Students might learn simple procedures from instruction, but this knowledge is often fragile (Merrill & Reiser, 1993). Ideally, students need to have more than rote procedures for solving problems — they need to have a causal understanding of why particular sequences of actions are effective (cf., Ohlsson & Rees, 1991).

Merrill and Reiser (1993) developed a theory of problem solving and learning that described understanding in a domain in terms of the *process in the world* that captures the structure and causality of a domain. Mastery of the process in the world enables novices to explain why some event took place. To be most effective, students must envision and reason about the situations represented in a problem rather than simply attempt to construct a sequence of actions that produces an answer. For example, arithmetic word problems may require reasoning about the connections between the mathematical relationships and real world situations (Fuson & Willis, 1988). This type of understanding allows students to constrain their problem solving search to reconstruct procedures when they are forgotten or confused and recover from errors more readily (Payne, 1988).

Table 1: The principles embodied in reasoning-congruent learning environments

- Render invisible behavior visible.
  - Make students' own reasoning explicit by having them make predictions of behavior.
  - Render behavior visible by allowing student to access normally invisible states.
- Support incremental planning and use of environment as note pad.
  - Minimize the translation process from the students' internal plans to the external representation of the solution.
  - Have the structure of a partial solution remind the students of where they are in their solution plan and the search space of the domain.
  - Allow students to focus on subproblems on the way to solving the entire problem, thereby avoiding premature commitment and exploiting independence of subgoals.
- Lead students to engage in more effective strategies.
  - Proactively guide problem solving by encouraging students to use a more profitable set of tools for solving problems.

Processes in the world are hard to acquire because they are often invisible or incompletely accessible (Collins, 1990). Mastery of the process in the world arises from cognitive activities novices perform to render it visible and to elaborate causes and effects. These activities might include prediction and explanation of future states (cf., Chi, Bassok, Lewis, Reimann, & Glaser, 1989), notation of future problem solving goals (cf., Singley, 1990), and connecting the formal notations with the situation in the world the problem embodies (Fuson & Willis, 1988; Nathan, Kintsch, & Young, 1992; White, 1993). The use of such appropriate strategies, called *solution processes* enables students to acquire the necessary mastery of the process in the world (Merrill & Reiser, 1993).

Merrill and Reiser (1993) argued that problem solving environments called *reasoning-congruent learning environments* can support novices acquiring effective solution strategies and support their access to and under-

standing of the process in the world. Merrill and Reiser (1993) presented six pedagogical goals of reasoning-congruent learning environments (see Table 1), and discussed several environments that exhibit one or more of these principles. Most of these environments are graphical in nature, such as a graphical geometric proof diagram (Koedinger & Anderson, 1993), a diagrammatic system for categorizing arithmetic word problems (Fuson & Willis, 1988), and a graphical LISP programming tutor (Reiser, Kimberg, Lovett, & Ranney, 1992). This might lead one to wonder if any benefits of reasoning-congruent learning environments are simply due to the graphical nature of the external representation used by students. Indeed, some of our principles are more effectively achieved using properties of graphical notations, such as spatial organization of the workspace. In fact, many designers of learning environments and other computer systems have assumed that graphical user interfaces make learning easier for novices. However, careful examinations of the outcomes of various graphical representations such as flowcharts for programming (Shneiderman, 1980) or data flow languages for logic (Green, Bellamy, & Parker, 1987; Green, Petre, & Bellamy, 1991) have found that graphics per se does not lead to higher performance on various tasks including comprehension or creation of objects. In our view, a graphical representation is not of itself sufficient to create a more effective learning environment. Instead, the critical issue is structuring the interactions to provide the guidance and support for the cognitive activities effective for problem solving in the domain.

In this paper, we describe an experiment designed to test the effectiveness of learning environments based upon the reasoning-congruence principles. We examine novices learning to program in LISP from one of three environments that differ in the degree to which they achieve the principles of reasoning-congruent learning environments. We will also attempt to disentangle some of the benefits of reasoning-congruence from the issue of graphical versus text-based representations. We expect environments that are more effective in implementing the principles of reasoning-congruent learning environments to help students more easily master the target domain, independent of whether the interface is graphical or text-based. In the next three sections, we describe the learning environments used in this study.

Description of the GIL environment

GIL (*Graphical Instruction in LISP*) is an interactive learning environment that helps students learn programming using a diagrammatic representation designed according to the principles of reasoning-congruent learning environments (Reiser et al., 1992). The first section of the GIL curriculum concerns basic list manipulation functions. It is important to lead students to focus on the behavior of each individual operator in a solution, so they can understand how each operator contributes to the final outcome (Merrill & Reiser, 1993). One useful way of focusing students' attention upon solution operators is to encourage them to make predictions of the operator's expected behavior, thereby highlighting knowledge gaps when a prediction is incorrect. In the *intermediate products* curriculum in GIL, we require students to make predictions of the input and output values of each step in a solution. Upon request, GIL tests these inferences. Students can ask GIL to apply their partial or complete solution to the input data. GIL evaluates the student's solution graphically, highlighting each function as it is examined, darkening each link as the data passes along it to the next function, and indicating any errors found.

The next phase of the GIL curriculum introduces variables in addition to arithmetic functions, logical functions, and predicates. Students now build solutions using variables that can take a variety of different values instead of working with specific data values. Students still build reasoning chains connecting functions to achieve some output, but students do not record intermediate values while constructing the solution. The important predictions of expected behavior now occur whenever students test a program. When a student tests a program, GIL asks for a value for each variable, and then asks the student to predict the solution's output with those inputs. After the test is completed, GIL reminds the student of the prediction, thereby facilitating explanation of incorrect predictions, and allows students access to intermediate states via a virtual probe that can inspect values input to or output from any function.

The next portion of the GIL curriculum concerns conditional processing, performing different actions depending upon the values of tests. In traditional LISP, these tests and actions are indicated by ordering within parentheses. In GIL they are indicated by test and action boxes into which students build programs just as they did in the variables portion of the curriculum.

GIL's notation, shown in Figure 1a, is designed to achieve all six principles of reasoning-congruent learning environments. We predict that this will lead GIL students to better understand the behavior of objects in the domain, and thus exhibit better problem solving than other environments that do not achieve the principles as completely.

Description of the SE environment

In contrast to GIL's graphical notation and the problem solving behaviors centered around the use of this notation, the second learning environment, SE (*Structured Editor*) uses the traditional text form of the language (Figure 1b). One of the major difficulties students face in learning to program concerns the need to focus on the syntax of expressions while trying to plan a solution. SE, like other syntax-directed editors, scaffolds some of the syntactic operations that are difficult for novices. In SE, students construct solutions by selecting functions, variables, or constants from a menu. SE constrains students to place functions and variables only in legal locations, providing cues to those locations as students drag the cursor across components of the program. SE also requires students to place the right parenthesis of a function call in a legal location, thus preventing un-

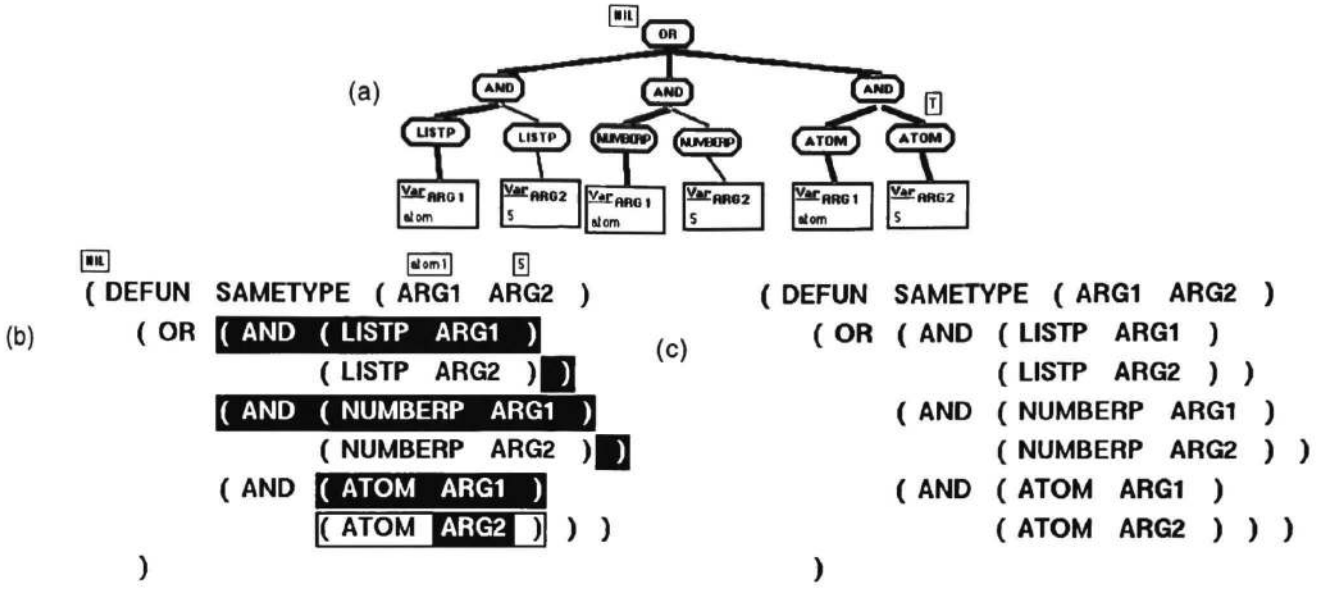

Figure 1: Examples of a problem in all three learning environments used in the study. (a) A test of a GIL solution, with a prediction and a use of the probe (on the right side). (b) A student working on the same problem in VSE, testing a solution. Already evaluated portions of the solution are darkened, and the current expression is boxed. (c) A completed solution from SE.

balanced parentheses. Functions and their arguments are not treated as a sequence of characters, but rather as unitary objects. Students can perform any editing operation upon an individual function or a function plus its arguments with a single key click, rather than operating on individual characters. SE also automatically redisplays the students' solutions using indentation to reflect the current structure, a technique known as pretty printing. Finally, SE was designed to allow students to work outside-in (placing new functions as arguments to current functions) or inside-out (wrapping new functions around current functions) as desired.

SE also allows students to test their solutions. Since SE students use variables throughout the curriculum, they enter values for each variable, as described in the GIL section above. Then SE provides the final output of the program. Note that SE students are not required to predict the outcomes of the test, which may result in SE students not doing so spontaneously, and thus having more difficulty identifying errors in the solution.

SE is designed to achieve two of the six principles of reasoning-congruent learning environments. First, it minimizes the translation between the way students plan their solutions and how they can be expressed by minimizing the need to focus on syntactic details of a plan's implementation. It also provides some support for students to work on subproblems independently. Indeed, in earlier pilot work we found that SE enabled students to learn LISP more easily than a traditional programming text editor. However, GIL is designed to implement more of the reasoning-congruent principles than SE, and in fact we believe GIL is even more effective on the two principles also present in SE, minimizing translation from plans to solutions and enabling work on subproblems. Thus, we expect that GIL students will master the curriculum with more understanding, and SE students will exhibit more difficulty planning their solutions and recovering from errors.

Description of the VSE environment

The third condition, VSE (*Visual Structured Editor*) is a modified version of SE that is designed to achieve more of the principles of reasoning-congruence than SE, and was designed to investigate the extent to which we could construct a reasoning-congruent learning environment within a text-based environment. Any learning advantages for students using VSE over students using SE must be due to the principles of reasoning-congruent learning environments that VSE achieves, and not from any differences in graphical and text notations.

VSE, like GIL, is designed to allow students access to invisible behavior and intermediate states. This principle is exhibited in the central contrast between the behavior of VSE and SE during students' tests of their programs. Like GIL, VSE asks students to predict the outcome of each test (Figure 1c). VSE displays the path of execution of a program during a test, by flashing and then highlighting each function as it is processed. This allows students to follow the order of evaluation. Further, students can probe the data passed from function to function, thereby rendering the solution's invisible behavior visible.

VSE is designed to achieve five of the six principles as completely as we could implement them in a text-based system. VSE students make predictions, have access to invisible states, lessened translation loads, the ability to work on subproblems, and are led to engage in useful ac-

tivities such as prediction of outcomes. However, as suggested earlier, we believe some principles of reasoning-congruent learning environments can be more effectively achieved using the spatial layout of a graphical representation. Therefore, we expect the performance of VSE students to be between SE and GIL.

In summary, this experiment compares three environments that differ in how effectively they achieve the principles of reasoning-congruent learning environments. As an environment more completely attains these principles, it should lead students to attain the understanding of the process in the world that needs to underlie students' procedures.

Method

Subjects. Thirty Northwestern University undergraduates (eight men and 22 women) took part in this study. All subjects had less than one college course of programming experience. Subjects were recruited through campus advertisements and paid $5 per hour. Subjects were randomly assigned to a learning environment so as to balance approximately Math SAT across conditions. Mean Math SATs were 623, 639, and 618 for SE, VSE, and GIL subjects respectively.

Materials. Subjects solved 22 problems divided into three sessions, covering the topics of list manipulation, logical functions, predicates, and conditional processing. The textbooks used in the three sessions amounted to roughly 50 pages of text, and were written in the notation (diagrammatic or textual) students used to solve problems. A pen-and-paper posttest contained four program construction problems, similar in difficulty to those in the learning sessions, and two debugging problems requiring students to determine whether a given program behaved correctly and, if not, to give an example of input data for which the program would behave incorrectly. The posttest used the notation the students had used during problem solving. Finally, following the first and third sessions students completed an evaluation questionnaire to assess their attitudes toward the domain and their performance (Reiser, Copen, Ranney, Hamid, & Kimberg, 1994).

Procedure. The three sessions of the experiment were spread over three to nine days, with no more than two days elapsing between sessions. Students worked independently in a cubicle with a computer. Students first read the initial portion of the textbook introducing LISP functions. After reading the first brief section, the experimenter demonstrated the learning environment for that condition. Students then continued solving problem sets interspersed with sections of the textbook. Students were free to take as long as they wished to solve the problems, but were required to work on each problem until the solution was correct. The learning environments recorded every student action and the time at which it took place, creating a behavioral protocol of all student problem solving events.

Results and Discussion

In this section, we address whether learning environments that conform to the principles of reasoning-congruent learning environments did in fact facilitate their students' problem solving, by analyzing a variety of measures taken during the learning sessions that indicate difficulty students experienced while solving the problems. Recall that the three learning environments, SE, VSE, and GIL, are ordered according to the degree to which the environment achieves the principles of reasoning-congruent learning environments. We predict that as the degree of reasoning-congruence increases, student pedagogical outcomes will improve. To test this prediction, we employed a linear planned comparison that tested an increase in effectiveness of problem solving ordered from SE to VSE to GIL. The central problem solving measures are summarized in Table 2.

First we considered the time required for students to complete the problems during the learning phase, called *solution time*. This is a rough measure of the amount of difficulty students had with the problems. As expected, as the environments increased in reasoning-congruence, students required less time to complete all problems. Also, notice that this difference heavily underestimates the differences between conditions, because tests in GIL and VSE, with their required predictions and graphics routines, took much longer than tests in SE.

Another measure of problem solving difficulty is the number of program elements deleted, because the deleted elements represent incorrect or unneeded portions of a solution that had to be removed. Again, as the environments increased in reasoning-congruence, subjects required fewer deletes, indicating more focused problem solving as a result of the reasoning-congruence of the environment.

One potential concern in evaluating the delete data is that it is possible in SE and VSE to delete multiple objects at once (by deleting a function and its arguments), while in GIL students must separately delete each object they wish to remove. Thus, perhaps the reasoning-congruence effect in number of deletes arises solely because it was easier to delete large amounts of the code in SE and VSE than to try to edit what was already on the screen. If so, then more deletes may indicate differences in editing styles rather than differences in the number of program components that required removal. To address this, we counted the times that students began deleting new things, regardless of how many objects were ultimately deleted in each sequence of deletes. We defined these *delete episodes* as one or more contiguous steps consisting of deletions of objects in the solution. Students who experienced more difficulty should exhibit more times during the problems in which they need to delete part of their programs. As expected, as the environments increased in reasoning-congruence, students exhibited fewer episodes in which they deleted parts of their programs.

Next we considered submitted answers during the acquisition session and the posttest. Again, we found decreasing problem solving difficulty with increasing

Table 2: Mean Values for the Problem Solving Measures on the Acquisition Session, Posttest, and Evaluation Questionnaire

Problem Solving Measure	*SE*	*VSE*	*GIL*	*F value*
Solution Time (min)	247	240	204	$F(1,26) = 19.4, p < .01$
Objects Deleted	189	157	103	$F(1,26) = 10.2, p < .01$
Delete Episodes	99	68	49	$F(1,26) = 8.99, p < .01$
Submissions Per Problem	4	2	2	$F(1,26) = 4.3, p < .05$
Errors on Posttest	13	17	11	$F(1,24) = 3.4, ns$

reasoning-congruence of the environment, as measured by the number of incorrect answers submitted during the learning session. Although there were many behavioral differences between the three conditions, there were no significant posttest differences, measured by the minimum number of edits required to repair each answer. Anderson (1983) argued that domain mastery arises out of solving problems correctly. Since all students in this study solved all problems correctly, it is not unexpected that there were no posttest differences. However, the behavioral differences showed that as students' environments became progressively more reasoning-congruent, they experienced less difficulty attaining an equivalent level of mastery.

Taken together, these results indicate that the reasoning-congruence of the environment facilitated students' problem solving. Indeed, even overall solution time decreased with reasoning-congruence, despite the more time-consuming nature of the procedure for testing programs (e.g., specifying predictions, slower graphical display of the program's evaluation). Apparently this is time worth investing, because it leads to more focused debugging, as evidenced by the need to repair less of the programs and fewer incorrect programs submitted as answers.

Conclusions

These results demonstrate that environments that are more reasoning-congruent allow students to attain equivalent domain mastery with a large reduction in problem solving effort. The reduction in problem solving effort is due to the reasoning-congruent learning environment students' mastery of the process in the world — the understanding of the domain that underlies their procedures for solving the problems. Reasoning-congruent learning environments scaffold students' solution processes to enable them to attain this conceptual understanding, whereas traditional environments may result in fragile rote procedures that do not lead to as successful outcomes.

Acknowledgments

This research was sponsored by contracts MDA903-92-C-0114 to Northwestern University and MDA903-87-K-0652 and MDA903-90-C-0123 to Princeton University from the Army Research Institute, and by a Spencer Foundation Dissertation Year Fellowship to the first author. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies of these institutions.

References

Anderson, J. R. (1983). *The architecture of cognition.* Harvard University Press, Cambridge, MA.

Chi, M. T. H., Bassok, M., Lewis, M. W., Reimann, P., & Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science, 13,* 145–182.

Collins, A. (1990). Cognitive apprenticeship and instructional technology. In Jones, B. F., & Idol, L. (Eds.), *Dimensions of thinking and cognitive instruction,* pp. 121–138. Erlbaum, Hillsdale, NJ.

Fuson, K. C., & Willis, G. B. (1988). Teaching representational schemes for solving addition and subtraction word problems. *Journal of Educational Psychology, 80,* 192–201.

Green, T. R. G., Bellamy, R. K. E., & Parker, J. M. (1987). Parsing and Gnisrap: A model of device use. In Olson, G. M., Sheppard, S., & Soloway, E. (Eds.), *Empirical studies of programmers: Second workshop,* pp. 132–146. Ablex, Norwood, NJ.

Green, T. R. G., Petre, M., & Bellamy, R. K. E. (1991). Comprehensibility of visual and textual programs: A test of superlativisms against the 'match-mismatch' conjecture. In Koenemann-Belliveau, J., Moher, T. G., & Robertson, S. P. (Eds.), *Empirical studies of programmers: Fourth workshop,* pp. 121–146. Ablex, Norwood, NJ.

Koedinger, K. R., & Anderson, J. R. (1993). Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design. In Lajoie, S. P., & Derry, S. J. (Eds.), *Computers as cognitive tools.* Erlbaum, Hillsdale, NJ.

Merrill, D. C., & Reiser, B. J. (1993). Scaffolding learning by doing with reasoning-congruent learning environments. In *Proceedings of the Workshop in Graphical Representations, Reasoning and Communication from the World Conference on Artificial Intelligence in Education (AI-ED '93),* pp. 9–16 Edinburgh, Scotland. The University of Edinburgh.

Nathan, M. J., Kintsch, W., & Young, E. (1992). A theory of algebra-word-problem comprehension and

its implications for the design of learning environments. *Cognition and Instruction, 9*, 329–389.

Ohlsson, S., & Rees, E. (1991). The function of conceptual understanding in the learning of arithmetic procedures. *Cognition and Instruction, 8*, 103–179.

Payne, S. J. (1988). Methods and mental models in theories of cognitive skill. In Self, J. (Ed.), *Artificial intelligence and human learning: Intelligent computer-aided instruction*, pp. 69–87. Chapman & Hall, London.

Reiser, B. J., Copen, W. A., Ranney, M., Hamid, A., & Kimberg, D. Y. (1994). Cognitive and motivational consequences of tutoring and discovery learning. Tech. rep., The Institute for the Learning Sciences, Northwestern University, Evanston, IL.

Reiser, B. J., Kimberg, D. Y., Lovett, M. C., & Ranney, M. (1992). Knowledge representation and explanation in GIL, an intelligent tutor for programming. In Larkin, J. H., & Chabay, R. W. (Eds.), *Computer-assisted instruction and intelligent tutoring systems: Shared goals and complementary approaches*, pp. 111–149. Erlbaum, Hillsdale, NJ.

Shneiderman, B. (1980). *Software psychology: Human factors in computer and information systems.* Little, Brown, and Company, Boston.

Singley, M. K. (1990). The reification of goal structures in a calculus tutor: Effects on problem solving performance. *Interactive Learning Environments, 1*, 102–123.

White, B. Y. (1993). Thinkertools: Causal models, conceptual change, and science education. *Cognition and Instruction, 10*, 1–100.