Schvaneveldt, R. W. (Ed). (1990). *Pathfinder associative networks: Studies in knowledge organization.* Norwood, NJ: Ablex.

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge.* Los Altos, CA: Morgan Kaufmann.

Chapter 2

Student Modeling in the ACT Programming Tutor

Albert T. Corbett
John R. Anderson
Alison T. O'Brien
Carnegie Mellon University

The goal of this project is to model a student's changing knowledge state during cognitive skill acquisition. Our purpose is to predict students' performance levels and to implement a mastery-based learning environment that enables students to achieve satisfactory performance levels in a minimum period of time. The project assumes a production system model of skill knowledge in which if-then production rules associate problem states and goals with actions and consequences. Our diagnostic goal is to model the student's knowledge of these rules and to model changes in the student's knowledge state over time. The project focuses on students learning to write programs in LISP with the assistance of an intelligent computer-based programming tutor. In the following sections we describe the learning environment, the curriculum, the cognitive model, and the learning and performance assumptions. Finally, we discuss our evaluations of the model.

THE ACT PROGRAMMING TUTOR

Our interest in student modeling and individualized instruction grows out of our work with the ACT Programming Tutor and its antecedents over the past 9 years (Anderson, Boyle, Corbett, & Lewis, 1990; Anderson, Conrad, & Corbett, 1989; Anderson & Corbett, 1992; Anderson, Corbett, Fincham, Hoffman, & Pelletier, 1992). The ACT Programming Tutor is a practice environment in which students write short programs in LISP, Prolog, or Pascal. The LISP and Prolog modules are presently employed to teach a self-paced introductory programming course

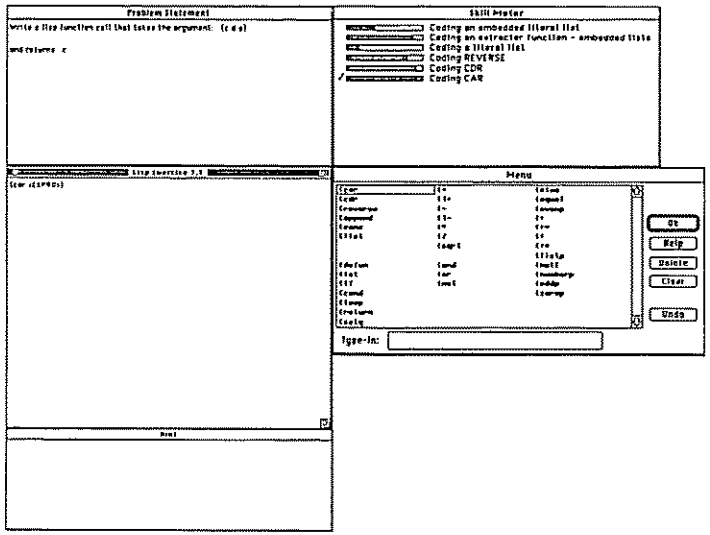


FIG. 2.1. The ACT Programming Tutor interface near the beginning of an exercise.

at Carnegie Mellon University and the Pascal module is used in an introductory programming course in a Pittsburgh high school.

Figure 2.1 depicts the tutor's screen display near the beginning of the first LISP exercise. The exercise description appears in the upper left window, and the student's solution appears in the code window immediately below. To complete this exercise the student needs to enter the LISP instruction *(car '(c d e))*. (The symbol *car* is a LISP operator and the expression *'(c d e)* is a list data structure on which *car* will operate.) The student enters code essentially a symbol at a time. In this figure, the student has just entered the operator *car*. The tutor has recognized this as a correct step and has expanded a template for this operator in the code window, including a goal node *<Expr0>* that represents the argument to the operator. The student will replace this goal symbol with an appropriate LISP expression, in this case the literal list *'(c d e)*.

Each of the three language modules is constructed around a set of several hundred language-specific rules for writing programs. This set of rules, termed the *ideal student model*, allows the tutor to generate exercise solutions (programs) step by step along with the student and provide feedback on the student's actions.

The tutor conventionally provides immediate feedback on each code symbol and requires any error be corrected before proceeding to the next step, so the student always remains on a recognized solution path.

We have been using such tutors to teach programming courses for the past 8 years and they have proven effective. Students generally work through the exercises more quickly with the tutor and perform as well or better on posttests (Anderson & Reiser, 1985; Corbett & Anderson, 1989, 1990, 1991; Corbett, Anderson, & Patterson, 1990). Despite this general effectiveness, however, some students flounder. This was particularly true the first time we taught LISP programming to novices in 1985. This experience led us to introduce student modeling into the tutor. As the student works the tutor maintains an estimate of the probability that the student has learned each rule in the ideal student model, in a process we call *knowledge tracing*. The skill meter in the upper right corner of Fig. 2.1 displays the tutor's model of the student that results from this process. Each programming rule that has been introduced into the curriculum is represented as a bar graph. The shading in the bar graph indicates the current estimate of the probability that the student knows the rule. In Fig. 2.1, six rules have been introduced. The probability that the student currently knows each rule ranges from .18 (the third rule down) to .99 (the rule at the bottom).

The goal of knowledge tracing is to implement mastery learning. The tutor continues presenting exercises in each section of the curriculum until the probability is very high that the student has learned each of the rules. (The rule at the bottom of the skill meter in Fig. 2.1 has reached this criterion and is checked as a consequence.) This knowledge-tracing procedure has passed a basic validity test; students perform better on posttest tests when knowledge tracing is in operation (Anderson et al., 1989). However, even with knowledge tracing in place, there is a wide range in posttest performance and some students struggle. The goal of the present research is to examine and improve the predictive validity of the modeling process. The following sections describe the production system cognitive model and the learning and performance assumptions that underlie knowledge tracing.

The Cognitive Model

The tutor was developed to test the ACT* theory of skill acquisition (Anderson, 1983). The theory distinguishes between declarative factual knowledge and procedural skill knowledge. Like many other cognitive theories (cf. Kieras & Bovair, 1986; Newell, 1990; Reed, Dempster, & Ettinger, 1985) ACT* assumes that procedural knowledge can be represented as a set of independent if-then productions rules that associate problem-solving states and goals to actions and their consequences. Skill acquisition consists of learning what actions satisfy a problem-solving goal, and what the consequences of the actions are for the problem-solving state, including the emergence of new goals. The LISP, Pascal, and Prolog

modules in the programming tutor each contain several hundred language-specific rules for writing programs that model performance essentially at the grain size of individual code symbols. For example, two productions are employed in coding the exercise displayed in Fig. 2.1. Recall that this exercise asks students to code a function call that returns *c* from the list *(c d e)* and the appropriate code is *(car '(c d e))*. The two productions that would fire are:

IF the goal is to return the first element of a list <list>,
THEN code a function call to *car* and set a goal to code <list>.

IF the goal is to code a literal list <literal-list>,
THEN code a single quote followed by <literal-list>.

The first production matches the goal that is set up by the problem description. When it fires it sets a subgoal that is satisfied in turn by the second production.

We call the set of programming rules built into the tutor the ideal student model, because it embodies the knowledge that the student is trying to acquire. This ideal model plays two roles in the tutor. First, it allows the tutor to solve exercises step by step along with the student in a process we call *model tracing*. As the student enters each code symbol, the tutor attempts to match the action to an applicable rule in the model. If a match is found, the tutor accepts the symbol and updates its internal representation of the problem state. If not, the tutor requires the student to try another action. Second, in knowledge tracing, the student is represented as an overlay of the ideal model (Goldstein, 1982). As the student works through the exercises, the tutor maintains an estimate of the probability that the student has learned each rule in the ideal model.

Our goal is to monitor the student's acquisition of such rules. When we analyze performance into a set of cognitive production rules and monitor performance across opportunities to apply the rules, we obtain rather orderly learning curves (see Fig. 2.3 later). This being the case, the question is whether we can monitor a student's performance and decide when we are satisfied that the student has learned the rule. For example, suppose a student has six opportunities to apply a rule and emits the following sequence of correct (1) and incorrect (0) actions:

1 0 1 0 1 1

What can we conclude about whether the student has learned this rule? More concretely, what is the probability that a student will perform the correct action at the next opportunity to apply the rule and how likely is the student to apply the rule correctly outside the tutor environment? Finally, how can we make these decisions efficiently online? One observation is that if this were a static assessment, our conclusion does not depend on the order of 1s and 0s. The sequences 0 0 1 1 1 1 and 1 1 1 1 0 0 lead us to the same conclusion as the sequence 1 0 1 0 1 1. However, we assume the student's knowledge state is changing as

a function of practice, so our estimate will be sensitive to the ordering of 1s and 0s.

The Learning and Performance Model

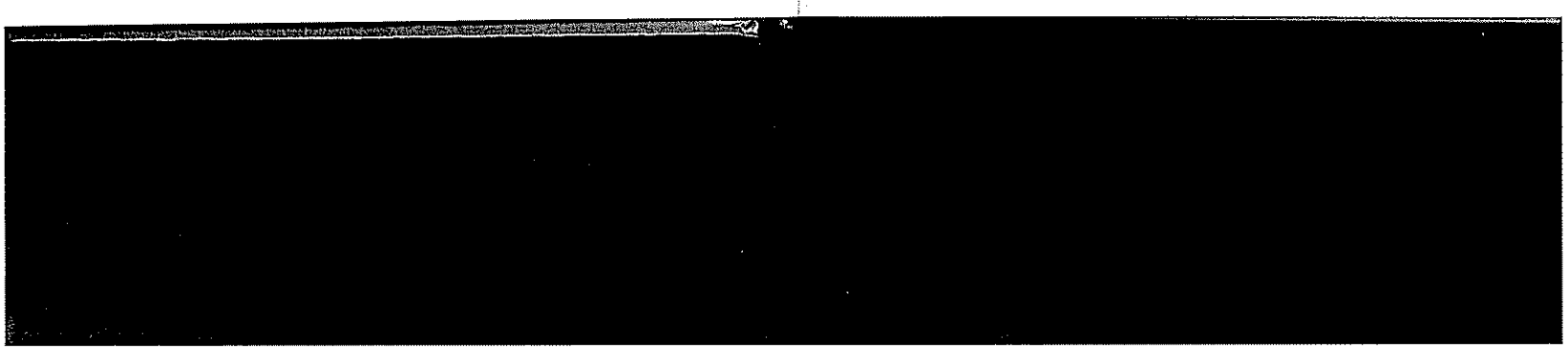
In addition to a cognitive model, knowledge tracing requires learning and performance assumptions. For the sake of simplicity, we have adopted a two-state learning model with no forgetting in knowledge tracing. In a two-state model, each coding rule is either in the learned or unlearned state. A rule can make the transition from the unlearned to the learned state at each opportunity to apply the rule, but rules do not make the transition in the other direction. We assume that performance is essentially determined by learning state, but recognize that there is some probability that if a rule is in the learned state, the student may slip and make a mistake and if the rule is in the unlearned state, the student may guess correctly. Within this framework, the task is to maintain an estimate of the probability that the student has learned each rule. Each time the student has the opportunity to apply a rule, the probability that the student knows the rule is updated, contingent on whether the student's action was correct or not. The Bayesian computational procedure used is a variation of one described by Atkinson (1972). This procedure employs two learning parameters and two performance parameters, displayed in Table 2.1. Following Atkinson, the probability p(T) of a transition from the unlearned to the learned state given an an opportunity to apply a rule is independent of whether the student applies the rule correctly or incorrectly. The computations for updating the knowledge state are described in the Appendix.

Mastery Learning

The knowledge-tracing process allows us to monitor students' knowledge state as they practice a cognitive task. The goal of knowledge tracing in the programming tutor is more ambitious, however. The tutor attempts to implement mastery learning by selecting appropriate exercises on the basis of the student model. Each lesson in the tutor is divided into sections in which a handful of program-

TABLE 2.1
The Learning and Performance Parameters in Knowledge Tracing

p(L$_0$)	Initial Learning	The probability a rule is in the learned state prior to the first opportunity to apply the rule (i.e., from reading the text).
p(T)	Acquisition	The probability a rule will make the transition from the unlearned to the learned state following an opportunity to apply the rule.
p(G)	Guess	The probability a student will guess correctly if a rule is in the unlearned state.
p(S)	Slip	The probability a student will slip (make a mistake) if a rule is in the learned state.



ming rules is introduced. In each section, the student reads an accompanying text that introduces the set of rules, then the tutor follows with a set of exercises that provides practice for the rules. The tutor continues to present exercises to the student until the probability that the student has learned each rule in the section has reached a criterion value of .95. Then the student moves on to the following section of text and exercises.

This procedure is analogous to individualized student-paced mastery learning systems based on Keller's Personalized System of Instruction (Block & Burns, 1976), although practice and assessment are usually treated as distinct phases in such systems. Such mastery learning systems have been shown to be successful in raising achievement scores (Block & Burns, 1976; Kulik, Kulik, & Bangert-Drowns, 1990; Kulik, Kulik, & Cohen, 1979). Verbal learning studies have shown that Bayesian remediation algorithms are successful (Atkinson, 1972; Atkinson & Paulson, 1972). When total practice is held constant, Bayesian remediation algorithms lead to higher posttest performance than does random selection of practice stimuli or student selection of practice stimuli. In this research we are interested in whether we can accurately predict students' performance in acquiring a knowledge-rich skill, by applying Bayesian estimation procedures to a production-rule model of the skill and bring students to mastery of that skill.

KNOWLEDGE TRACING: INTERNAL VALIDITY

As discussed earlier, the knowledge-tracing and remediation mechanism satisfied a weak validity test in its first use (Anderson et al., 1989). Students who completed a fixed set of required exercises followed by an individualized sequence of remedial exercises performed reliably better on posttests than students who completed only the required exercises. Recently we have begun examining the validity of knowledge tracing in more detail. In these studies we are interested in the absolute accuracy of the knowledge-tracing model in predicting performance.

Predicting Tutor Performance

The ultimate goal of knowledge tracing is to predict later programming performance outside the tutoring environment. However, our two initial assessments, Corbett and Anderson (1992a,1992b, 1993), focused on modeling students' performance with the tutor for two reasons. First, performance predictions can readily be studied in detail in an environment that imposes immediate feedback and correction and, second, internal validity is logically a prerequisite for predictive validity outside the tutoring environment. The model that underlies knowledge tracing allows us to predict accuracy at each goal (step) in the tutor exercises with the following formula:

$$p(C_{gs}) = p(L_{rs}) * [1 - p(S_r)] + [1 - p(L_{rs})] * p(G_r) \quad (1)$$

That is, the probability that a student s will perform an appropriate action at goal g is the sum of two products: (a) the probability that an appropriate rule r is in the learned state for student s times the probability of a correct response if the rule is in the learned state, and (b) the probability that an appropriate rule r is not in the learned state for student s times the probability of a correct guess if the rule is not in the learned state.

To assess internal validity, we compute error rate across students for each coding goal in the required tutor exercises, compute the error rate predicted by the knowledge-tracing model at each goal, and correlate these measures across goals. The first study (Corbett & Anderson, 1992b) demonstrated that holding the four learning and performance parameters $p(L_0)$, $p(T)$, $p(G)$, and $p(S)$ constant is inadequate for modeling behavior. Best fitting constant parameter estimates yielded a correlation of .58 across the 25 exercises (158 coding goals) in the curriculum. A much better fit is obtained when the four parameters are allowed to vary across programming rules, $r = .85$, mean error = .01, mean absolute error = .07. This fit is displayed in Fig. 2.2.

Learning Curves

In addition to the overall fit, we can evaluate the internal validity of the model by examining the learning curve for individual production rules. Figure 2.3 displays the mean learning curve across the 21 rules under investigation. On average,

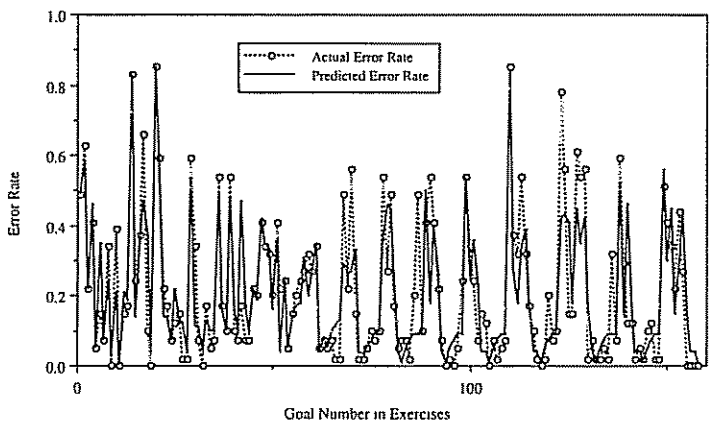


FIG. 2.2. Actual and predicted error rates across subjects at each goal in a set of required tutor exercises. From A. T. Corbett and J. R. Anderson, 1992b. Copyright 1992 by Springer-Verlag. Reprinted by permission.

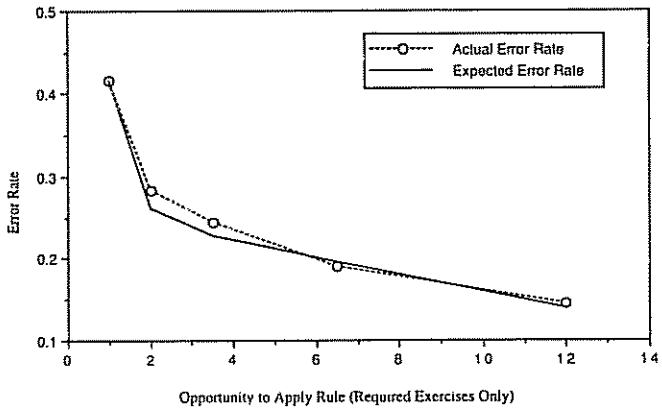


FIG. 2.3. Mean actual error rate and expected error rate for the ideal coding rules across successive rule applications.

error rate decreases monotonically with opportunity to exercise a rule, as the learning and performance models predict. However, some rules deviate substantially from the model predictions. Figure 2.4 displays the learning curve for one production rule that deviates pronouncedly from expected performance at two points. This pattern suggests that at least two distinct programming rules have been conflated in the model. No adjustment in the learning and performance parameters can capture this deviation; it represents an error in the cognitive model. We adjusted the cognitive model in light of such deviations, expanding the number of rules by 67% and examined the fit of the revised model in a second study. The tutor's parameter estimates were allowed to vary across productions and were derived as far as possible from the first-term curve fits. These built-in tutor parameters yielded a better fit to the data than in the previous term, $r = .71$. Again, best fitting parameter estimates were generated after the fact, yielding an improved fit, $r = .90$, mean error $= -.02$, mean absolute error $= .06$.

KNOWLEDGE TRACING: PREDICTING POSTTEST PERFORMANCE

More recently we have undertaken a study to evaluate how well the model predicts posttest performance. In this study, students completed posttests with a structure editor that is virtually identical to the tutor, except that no help is

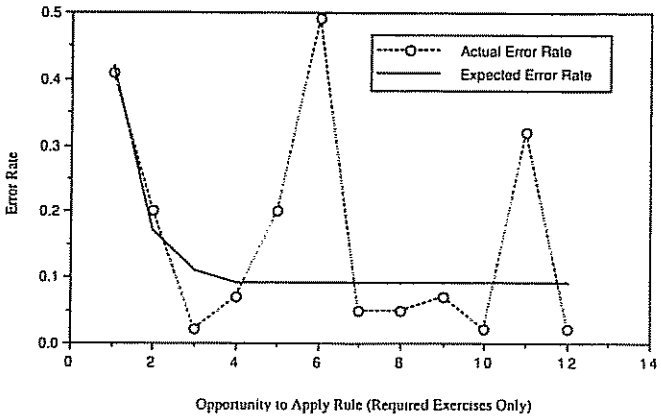


FIG. 2.4. Actual and expected error rate across successive opportunities to apply a single coding rule. From A. T. Corbett and J. R. Anderson, 1993. Copyright 1993 by Springer-Verlag. Reprinted by permission.

provided. This editor has three useful qualities: (a) Symbols (atoms) are the basic unit of code entry, as in the tutor, (b) the editor ensures that the student's code is syntactically well formed, and (c) the editor generates a complete trace of the students' actions.

This posttest environment should enable us to make absolute performance predictions at the level of whole exercises and at the level of individual goals in completing the posttest. An analysis at the level of individual posttest goals is of interest for two reasons. First, it represents a fine-grained evaluation of the predictive validity of student modeling in the tutor. Second, it affords us an opportunity to begin applying the knowledge-tracing model to environments in which students are not channeled onto a correct solution path through immediate feedback. We have implemented a number of nonimmediate feedback tutors in the past that have a variety of attractive features. Although the analysis becomes more complicated, it should be possible to generalize knowledge tracing to such environments.

The Curriculum

As in the two earlier validity studies, students worked through the early sections of the LISP curriculum in this study. Even the earliest sections of the curriculum pose serious difficulties for many students for at least three identifiable reasons. The surface structure of the language is the same as the surface form of data

structures, so students readily confuse symbols and referents. Second, students are unfamiliar with the recursive nature of functional evaluation. Finally, students have trouble grasping the structure of lists in LISP. Unlike everyday lists, lists in LISP have a hierarchical structure and list operations act on these structures hierarchically. As a result, students must struggle with the type of knowledge reformulation that has been observed in other domains (Carey, 1985; Larkin, McDermott, Simon, & Simon, 1980; Lesgold et al., 1988; McCloskey, 1983) and that poses a strong challenge to any student modeling efforts.

Twenty college students worked through the first six sections of the curriculum at their own pace. In each section students read text describing LISP then completed a set of required exercises that cover the programming rules being introduced. In the first five sections, students completed remedial exercises as needed to bring all production rules introduced in the section to a mastery criterion (minimum learning probability of .95). In the sixth section, students completed a required set of 12 exercises with no remediation. (The required set in this section represents twice the minimum number of exercises that would be required to bring all rules to criterion if no errors were made.) Students completed 64 required exercises across the six sections and an average of 18 remedial exercises. The number of remedial exercises ranged from 3 to 63 across students.

Following the first, fourth, and sixth sections students completed a posttest. The posttests contained 6, 12, and 15 programming exercises respectively. These exercises are selected from the same population as the tutor exercises.

Internal Validity

We used the knowledge-tracing model to predict accuracy at each goal (step) as students worked with the tutor. Students completed 345 coding goals across the 64 required tutor exercises (i.e., an average of just over five coding steps per exercise). At each of the 345 goals in the required exercises, we first predicted the probability of a correct response given the applicable rule, then applied the the knowledge-tracing procedure to update the learning probability for the applicable rule. We did not fit (predict accuracy for) the remedial exercises, however we did update learning probabilities at these goals, just as the tutor would. Best fitting estimates yielded a good fit to the data; actual and predicted accuracy values correlated .90, mean error = .001, mean absolute error = .04. We use the best fitting parameter estimates in predicting posttest performance next, so the results reflect an upper bound on the validity of the model.

Posttest Predictive Validity

In principle we can predict response accuracy at each goal in the posttest with Equation 1 just as in the tutor. We can use the final learning probability $p(L_n)$ computed for each rule and subject in knowledge tracing to predict whether

subject s will apply rule r successfully when needed in the posttest. In applying Equation 1 to the posttests, we assume the same performance parameter estimates p(S) and p(G) as in the tutor, and make the simplifying assumption that no learning is occurring [p(T) = 0].

Exercise Scores. If production rules are independent as the model assumes, then we can derive one straightforward accuracy prediction. The probability that a student will complete an exercise correctly is given by

$$\pi p(C_{rs}) \tag{2}$$

the product of the probabilities that the student will respond correctly at each successive goal in solving the exercise. For each student and each posttest we computed (a) the proportion of exercises actually completed correctly and (b) the mean predicted probability of completing the exercises correctly. The average of these statistics across subjects is displayed in Table 2.2. In addition, the correlation of the actual and predicted measures (r_{AP}) across subjects and the mean absolute error (MAE) of the predicted values is displayed. As can be seen knowledge tracing is predicting average performance quite well. The actual and expected accuracy values are quite similar across the three posttests. Students are performing somewhat better than expected in the second posttest and somewhat worse in the third posttest.

Note that knowledge tracing is completely insensitive to individual differences in the first two posttests. These two tests followed sections in which students had worked to mastery. Consequently there is little variability in learning probability estimates across students in these sections and no variability in performance predictions. In the final tutor section, however, students did not work to mastery. Absolute accuracy is substantially lower on the third test and, in this case, the knowledge-tracing model is sensitive to individual differences. The correlation of actual and expected values, $r = .69$, is significantly different from 0. (The sample size is 19, because one student's posttest data were lost. Correlations exceeding .47 are significantly different from 0 at the .05 level in this

TABLE 2.2
Actual and Expected Proportion of Exercises Completed Correctly Across Subjects in Each of the Three Posttests

	Mean Proportion Correct			
	Actual	*Predicted*	r_{AP}	*MAE*
Posttest 1	.89	.90	—	.09
Posttest 2	.89	.84	—	.11
Posttest 3	.55	.64	.69	.18

and following sections.) Under these circumstances, the learning probabilities are highly predictive of students' posttest performance.

Posttest Goal Accuracy. In attempting to predict accuracy at each goal in the posttest exercises, decisions must be made about how to score actions when a student has left a recognizable solution path. Two things are required: (a) a model of which programming goals are mutually independent, and (b) a plan for assimilating any intervening actions if the student leaves and returns to a recognizable solution path. We used a relatively simple scheme with three rules: (a) Sibling goals in the (hierarchical) goal tree are judged mutually independent, so if a student makes a mistake at one goal we continue tracing actions at sibling goals; (b) if a student makes a within-category error (substituting one constructor function for another or one extractor for another), we continued tracing descendent goals; and (c) if a student makes a noncategorical error, tracing of descendents is suspended, but resumed if the error is corrected. This scoring scheme is overly strict. If the student makes a single error, for example, omitting the outermost of nested extractor functions, or perhaps inverting the the order of nested extractor functions, each action in the series may be scored as incorrect.

For each student and each posttest we computed (a) the proportion of goals correctly satisfied across all exercises and (b) the mean expected probability of correctly satisfying a goal across exercises. The average of these statistics across subjects is displayed in Table 2.3. In addition, the correlation of the actual and expected measures across subjects (r_{AP}) and the mean absolute error of the predicted values (MAE) is displayed. The overall pattern of results is quite similar to Table 2.2. The actual and expected accuracy rates are quite similar, although students are underperforming expectations on the third posttest. Expected accuracy is invariant across students in the first two tests, but very strongly correlated with actual accuracy in the third test, r = .82.

The knowledge-tracing model was quite accurate at predicting average group performance. This characteristic alone would be of use in a training system designed to bring a group of students as a whole to a mean acceptable performance level. However, the model substantially underpredicted variability among students, particularly on the first two tests. Although the variability in the observed

TABLE 2.3
Actual and Expected Proportion of Goals Satisfied Correctly Across Subjects in Each of the Three Posttests

	Mean Proportion Correct			
	Actual	*Predicted*	r_{AP}	*MAE*
Posttest 1	.94	.95	—	.05
Posttest 2	.94	.95	—	.05
Posttest 3	.83	.93	.82	.11

TABLE 2.4
Correlation of Actual Posttest Accuracy With Number of Errors Made in the Fixed Set of Relevant Exercises in the Tutor Curriculum

	Exercises	*Rules*
Posttest 1	−.25	−.24
Posttest 2	−.64	−.75
Posttest 3	−.68	−.84

scores for the three posttests is not much different than would be expected by chance, we wondered if individual differences in posttest performance could be predicted from tutor performance. To examine this we correlated the number of errors made in completing the relevant required tutor exercises with actual posttest accuracy across students for each posttest. These correlations are displayed in Table 2.4.

As can be seen, the more errors a student made in completing the fixed set of tutor exercises, the less accurate the student was on the posttest. This correlation is nonsignificant for the first posttest as it should be, but large for the second and third posttest. The results for the second posttest pose a problem for knowledge tracing. All students have achieved mastery on the relevant rules according to the model and should be performing comparably. The number of errors made on the way to mastery should have no implications for posttest performance, but it does. (Because students did not achieve mastery on the rules in the final section, the correlation for the third posttest is not necessarily a problem.) There are several reasons why error rate might be correlated with performance on the second posttest; one concerns individual differences in parameter estimates and the second concerns individual differences in retention. We examined each of these two possibilities in the form of model revisions.

Individual Differences in Parameter Estimates

The learning and performance parameters in the basic model reflect rule difficulty, because they are allowed to vary across rules. However, they do not reflect individual differences among students. In using best fitting parameters for the group as a whole, we tend to underestimate the knowledge state of students who are doing well, but overestimate the knowledge state of students who are struggling. Consequently, students who are making few errors and completing few remedial exercises nevertheless tend to be overlearning whereas students making many errors and doing many remedial exercises still tend to be underlearning. This pattern would result in the observed negative correlation of tutor errors and posttest accuracy.

We revised the basic model to reflect individual differences by incorporating four individual difference weights for each student, one weight for each of the

four parameters types [$p(L_0)$, p(T), p(G), p(S)]. In fitting each student's data, we converted each of the group parameter estimates to odds form, multiplied by the corresponding subject-specific weight and converted the resulting odds back to a probability as shown here:

$$\frac{p_{ir} * w_{is}}{p_{ir} * w_{is} + (1 - p_{ir})} \quad (3)$$

In this formula, i indicates the parameter type [$p(L_0)$, p(T), p(G), p(S)], r the rule, and s the student. We derived a best fitting set of four weights across the 54 rules for each student. This in turn yields a set of 216 individualized parameter estimates for each subject. These parameter estimates were then used to refit the tutor data and predict posttest performance. The resulting posttest predictions are displayed in Tables 2.5 and 2.6.

As can be seen, the absolute accuracy predictions across exercises and rules are similar to those obtained earlier with group parameter estimates. However, the revised fit is considerably more sensitive to individual differences in the second posttest. The correlation of actual and expected exercise accuracy is .51 for the second test, whereas the correlation of actual and expected accuracy at the level of individual rules is .64.

TABLE 2.5
Individual Differences: Actual and Expected Proportion of Exercises Completed Correctly Across Subjects in Each of the Three Posttests, Based on Individualized Parameter Estimates

	Mean Proportion Correct			
	Actual	*Predicted*	r_{AP}	*MAE*
Posttest 1	.89	.91	−.02	.10
Posttest 2	.89	.86	.51	.10
Posttest 3	.55	.68	.72	.16

TABLE 2.6
Individual Differences: Actual and Expected Proportion of Goals Satisfied Correctly Across Subjects in Each of the Three Posttests, Based on Individualized Parameter Estimates

	Mean Proportion Correct			
	Actual	*Predicted*	r_{AP}	*MAE*
Posttest 1	.94	.95	−.01	.05
Posttest 2	.94	.95	.64	.05
Posttest 3	.83	.94	.80	.11

Individual Differences in Retention

There is a second reason that the model may underpredict individual differences in posttest performance. There may be differences not just in learning and performance parameters, but also in retention. Note that the "retention interval" between learning and test is not held constant across students in mastery learning. Students who make more errors and complete more exercises experience longer retention intervals filled with related and potentially interfering activity. Consequently, students who make more errors may also experience more forgetting prior to each test.

A second interesting retention issue concerns the rules that students learn. In principle the student model consists of ideal programming rules. In knowledge tracing, the problem of rule identifiability quickly arises, however. Students may acquire suboptimal rules that lead to perfect performance in some contexts and are only exposed in other contexts (VanLehn, 1990). A simple example involves the ideal rule for declaring variables in LISP programs: *Declare one variable in a function definition for each argument that will be passed when the function is called.* For example, a function that takes three expressions and puts them in a list should have three variables. Figure 2.5 displays the learning curve for this ideal rule. As can be seen, error rate decreases monotonically as expected over the first 4 opportunities to apply the rule, then rises for 5th and 6th opportunities and again to a lesser extent at the 10th opportunity. The first four points reflect

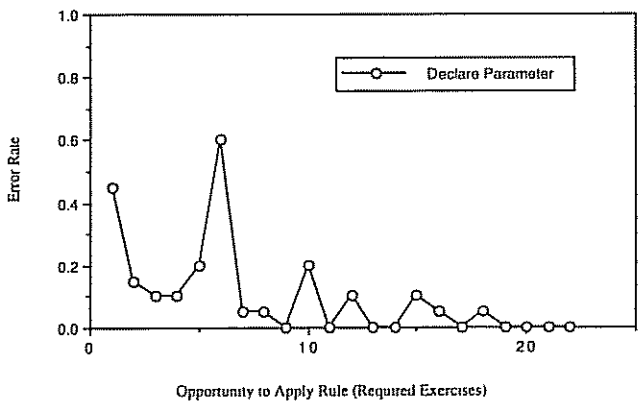


FIG. 2.5. Error rate in successive opportunities to apply the hypothetical rule that declares a variable in a function definition.

four exercises that each require a single variable declaration. The 5th and 6th and 9th and 10th opportunities represent the first two exercises in which two variables must be declared. Judging from the error rate at the sixth opportunity, more than half the students initially formed some suboptimal rule for declaring variables over the first four exercises. That rule might have been overly general, that is, *always declare one variable*, or overly specific, that is, *if there is one argument then declare one variable*. In either case, the student is unprepared to deal with a program that takes two arguments.

One solution to this identifiability problem would be to estimate probabilities that the student has learned the ideal rule and each of the competing suboptimal rules. However, we have adopted an alternative solution that monitors the following two rules: (a) Declare one variable for the first argument in a function definition, and (b) declare subsequent variables for subsequent arguments. Figure 2.6 shows the learning curves for these two rules. Note that this approach does not attempt to discriminate among the ideal rule and various suboptimal rules that students might be applying in the first context. Instead, the tutor monitors performance separately in the two contexts that will reveal suboptimal rules. In this sense it is more accurate to the say the tutor is monitoring contexts rather than rules. This approach has the advantage that it is computationally simple and effective practically. If students demonstrate mastery in each of the relevant contexts, then they should be able to perform programming tasks as if they had acquired the optimal rule.

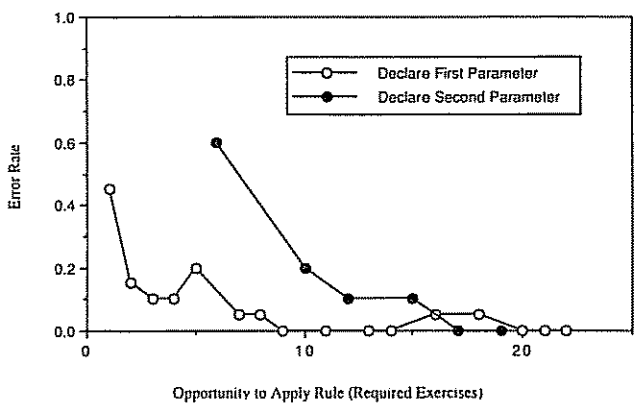


FIG. 2.6. Error rate in successive opportunities to apply two hypothetical rules that declare a variable in different contexts.

The exact cognitive rules that students form may have implications for retention, however. Students who formulate the correct rule in the first place have 22 opportunities to apply it in the previous figures. Students who initially formulate suboptimal rules get less practice on whatever rules they eventually form. Because suboptimal rules are ultimately revealed as errors, this may explain why error rates in the tutor correlate negatively with posttest performance.

To approximate retention effects we implemented a simple interference model of retention, again following Atkinson (1972). We assume that each time a student makes a mistake in applying a rule, there is a small probability that the student will forget any of the other rules that have been introduced up to that time. More concretely, each time the student makes a mistake applying a rule, the probabilities that the student has learned the other rules are each multiplied by .995. This model literally assumes that forgetting results from the learning processes associated with errors, but also generally captures the expected relationship between errors and forgetting in mastery learning. We used this forgetting model and the group learning and performance parameters to refit the tutor data and predict posttest performance. The resulting posttest predictions are displayed in Tables 2.7 and 2.8. As can be seen the absolute accuracy predictions across exercises and rules is similar to that achieved with the group parameter estimates (cf. Tables 2.2 and 2.3). However, the revised fit is considerably more sensitive to individual differences on the second posttest than are those earlier fits. In par-

TABLE 2.7
Individual Differences in Retention: Actual and Expected Proportion of Exercises Completed Correctly Across Subjects in Each of the Three Posttests, Based on Individualized Parameter Estimates

	Mean Proportion Correct			
	Actual	*Predicted*	r_{AP}	*MAE*
Posttest 1	.89	.90	.43	.09
Posttest 2	.89	.80	.59	.13
Posttest 3	.55	.52	.75	.16

TABLE 2.8
Individual Differences in Retention: Actual and Expected Proportion of Goals Satisfied Correctly Across Subjects in Each of the Three Posttests, Based on Individualized Parameter Estimates

	Mean Proportion Correct			
	Actual	*Predicted*	r_{AP}	*MAE*
Posttest 1	.94	.95	.10	.05
Posttest 2	.94	.93	.65	.06
Posttest 3	.83	.91	.85	.09

ticular, the correlation of actual and predicted accuracy for the second test is .59 for exercises and .65 at the level of individual rules.

Individual Posttest Exercises

The two variations on the basic model that accommodate individual differences in learning and performance and that reflect retention effects both fit the overall data well. We can examine the two models in more detail by comparing their performance predictions for individual exercises. Figure 2.7 displays actual performance on each of the 15 exercises in Posttest 3 along with the predictions of the two revised models. The first seven exercises, labeled "new" exercises, are drawn from the same population as the final two tutor sections that immediately preceded the test. The final eight exercises, labeled "old" are characteristic of the first four tutor sections. Accuracy levels are considerably higher for the old than for the new exercises in this figure, because the old exercises are intrinsically easier. Performance levels on these eight old exercises has dropped substantially compared to the second posttest, however.

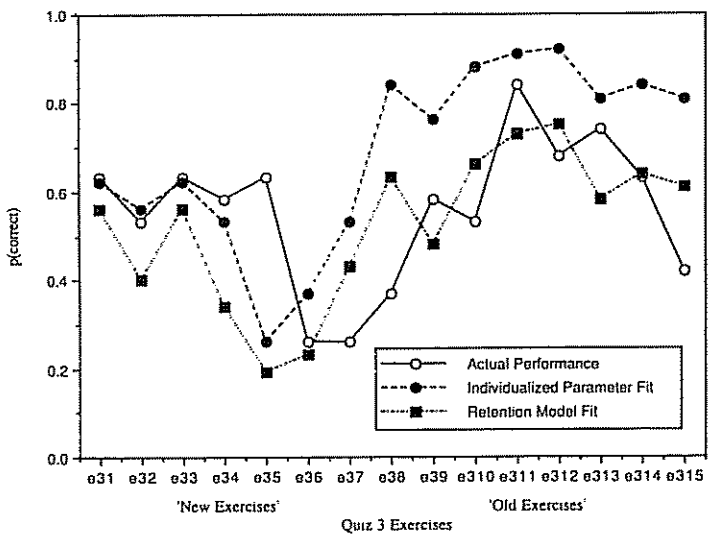


FIG. 2.7. Actual accuracy and the predictions of two models for the 15 exercises in the third posttest. Filled circles represent the model in which the four learning and performance parameters are individualized. Filled squares represent the model that incorporates forgetting.

As can be seen, the predictions of the two models follow the contours of the data fairly well, but the goodness of fit of the two models interacts with exercise type. The model with individualized learning and performance parameters fits the new exercises reasonably well, but overpredicts accuracy on the old exercises. This pattern is to be expected in the absence of any forgetting assumptions. Meanwhile, the model that incorporates a retention factor fits the old exercises quite well, but underpredicts performance on the new exercises. This pattern suggests that the forgetting assumptions are too simple and could have various implications. One possibility is that there are differential interference effects for rules that are actively being exercised (e.g., in the current section of the curriculum) and those that are not. A second possibility is that there is a purely temporal forgetting factor in addition to specific interference effects.

CONCLUSIONS

Our effort to predict students' performance based on a production rule model of knowledge and simple learning and performance assumptions has proven generally successful. The basic knowledge-tracing model predicts average performance quite well, both for the tutor and for a conventional programming environment in which students work without feedback. The basic model is also quite sensitive to individual differences, as demonstrated in the third posttest, but loses its sensitivity to genuine differences as learning probability estimates rise to the mastery criterion. Two variations on the basic knowledge-tracing model, however, are each shown to be quite effective in modeling students' posttest performance, both at the level of complete exercises and individual rule firings, even as learning probabilities rise. One of these models incorporates individual differences in learning and performance in the form of four multiplicative weights per student. The second model incorporates retention effects in the form of a multiplicative forgetting factor. Each of these revised models shows some systematic deviations from the performance data, however. The former model systematically overpredicts performance on old exercises whereas the latter underpredicts performance on new exercises. Deploying the two variations simultaneously may serve to minimize these deviations. Each of these variations requires development for optimal deployment, however. In exploring the former model, the four multiplicative weights were derived for each student by fitting the student's complete body of work with the tutor. This estimation procedure does not lend itself directly to dynamic modeling as the student works. Instead, this model requires an additional procedure for estimating individualized student parameters as the student works. The latter model, reflecting retention effects, can be readily implemented in a dynamic performance environment, although additional time or interference assumptions need to be incorporated, as suggested earlier.

Beyond refining the individualization of learning and retention, a major goal is to extend the knowledge-tracing process to other environments. The program-

ming tutor conventionally constrains the student to a recognizable solution path through immediate error feedback and correction. However, the fit of the model to posttest performance suggests that the model can be generalized to other performance environments. We have developed variations on the tutoring environment that allow students to deviate from recognizable solution paths. One version provides no help until asked by the student (Corbett et al., 1990). A second version volunteers immediate information on errors but does not require immediate correction (Corbett & Anderson, 1989). Both versions ensure that the student eventually codes a working solution. We should be able to apply the knowledge-tracing model in each of these environments, much as we did in predicting posttest performance, to update the student's knowledge state as well as to predict performance.

In general, it should be possible to extend knowledge tracing to any practice environment. Tutorial feedback is not required, although the student must have some standard for assessing his or her actions. What is required is a cognitive model of the task that associates problem states and goals with both actions and their consequences for the problem state. Immediate error feedback simplifies this task, because it is sufficient to model correct actions at each goal. Eliminating immediate error feedback and, for that matter, the assurance that the student will reach a correct solution, requires that the model be extended to incorporate incorrect actions. The success of the model in predicting performance in the posttest environment suggests that relatively simple assumptions make this a tractable task.

ACKNOWLEDGMENT

This research was sponsored by the Office of Naval Research under Grant N00014-91-J-1597.

APPENDIX

Each time the student has the opportunity to apply a rule, we need to estimate the probability that the rule is in the learned state, contingent on the accuracy of the student's response.

The probability $p(L_n/C_n)$ that a production has been learned following a correct action at the nth opportunity to apply the rule is shown in Equation 1. This probability is the sum of two probabilities: (a) the probability that the rule was already in the learned state, given a correct action, and (b) the probability the rule enters the learned state at this opportunity, if it was not already there. The first of these probabilities expands as Bayes' theorem as shown in Equation 3.

Similarly, the probability $p(L_n/E_n)$ that the production has been learned following an error at the nth opportunity to apply a rule is expressed in Equation 2. This probability is also the sum of two probabilities: (a) the probability that the rule was already in the learned state, given an error, and (b) the probability the rule enters the learned state at this opportunity, if it was not already there. The first of these probabilities also expands as Bayes' theorem as shown in Equation 4.

Equations

$$p(L_n/C_n) = p(L_{n-1}/C_n) + [1 - p(L_{n-1}/C_n)] * p(T) \quad (1)$$

$$p(L_n/E_n) = p(L_{n-1}/E_n) + [1 - p(L_{n-1}/E_n)] * p(T) \quad (2)$$

$$p(L_{n-1}/C_n) = p(L_{n-1}) * p(C/L) / [p(L_{n-1}) * p(C/L) + p(U_{n-1}) * p(C/U)] \quad (3)$$

$$p(L_{n-1}/E_n) = p(L_{n-1}) * p(E/L) / [p(L_{n-1}) * p(E/L) + p(U_{n-1}) * p(E/U)] \quad (4)$$

Definitions

$p(L_n)$	The probability a production rule is in the learned state following the nth opportunity to apply the rule.
$p(L_{n-1})$	The probability a production rule is in the learned state prior to the nth opportunity to apply the rule.
$p(U_{n-1})$	The probability a production rule is in the unlearned state prior to the nth opportunity to apply the rule.
C_n	A correct action at the nth opportunity to apply a rule.
E_n	An error at the nth opportunity to apply a rule.
p(C/L)	The probability of a correct action if the rule is currently in the learned state.
p(E/L)	The probability of a slip (an error) if the rule is currently in the learned state.
p(C/U)	The probability of a correct guess if the rule is currently in the unlearned state.
p(E/U)	The probability of an error if the rule is currently in the unlearned state.
p(T)	The probability a rule will transit from the unlearned to learned state given the opportunity to apply the rule.

REFERENCES

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

Anderson, J. R., Boyle, C. F., Corbett, A. T., & Lewis, M. W. (1990). Cognitive modelling and intelligent tutoring. *Artificial Intelligence, 42*, 7–49.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP Tutor. *Cognitive Science, 13*, 467–505.

Anderson, J. R., & Corbett, A. T. (1992). Acquisition of LISP Programming Skill. In S. Chipman & A. Meyrowitz (Eds.), *Foundations of knowledge acquisition: Cognitive models of complex learning* (pp. 1–24). Hingham, MA: Kluwer.

Anderson, J. R., Corbett, A. T., Fincham, J. M., Hoffman, D., & Pelletier, R. (1992). General principles for an intelligent tutoring architecture. In V. Shute & W. Regian (Eds.), *Cognitive approaches to automated instruction* (pp. 81–106). Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J. R., & Reiser, B. J. (1985). The LISP Tutor. *Byte, 10*(4), 159–175.

Atkinson, R. C. (1972). Optimizing the learning of a second-language vocabulary. *Journal of Experimental Psychology, 96*, 124–129.

Atkinson, R. C., & Paulson, J. A. (1972). An approach to the psychology of instruction. *Psychological Bulletin, 78*, 49–61.

Block, J. H., & Burns, R. B. (1976). Mastery learning. *Review of Research in Education, 4*, 3–49.

Carey, S. (1985). *Conceptual change in childhood.* Cambridge, MA: MIT Press.

Corbett, A. T., & Anderson, J. R. (1989). Feedback timing and student control in the LISP Intelligent Tutoring System. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Artificial intelligence and education: The proceedings of the 4th International Conference on AI and Education* (pp. 64–72). Springfield, VA: IOS.

Corbett, A. T., & Anderson, J. R. (1990). The effect of feedback control on learning to program with the LISP Tutor. In M. Piattelli-Palmarini (Ed.), *Proceedings of the Twelfth Annual Conference of the Cognitive Science Society* (pp. 796–803). Hillsdale, NJ: Lawrence Erlbaum Associates.

Corbett, A. T., & Anderson, J. R. (1991). *Feedback control and learning to program with the CMU LISP Tutor.* Paper presented at the annual meeting of the American Educational Research Association, Chicago.

Corbett, A. T., & Anderson, J. R. (1992a). Knowledge tracing in the ACT Programming Tutor. In J. Kruschke (Ed.), *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society* (pp. 623–628). Hillsdale, NJ: Lawrence Erlbaum Associates.

Corbett, A. T., & Anderson, J. R. (1992b). Student modeling and mastery learning in a computer-based programming tutor. In C. Frasson, G. Gauthier, & G. McCalla (Eds.), *Intelligent tutoring systems: Second international conference on intelligent tutoring systems* (pp. 413–420). New York: Springer-Verlag.

Corbett, A. T., & Anderson, J. R. (1993). Student modeling in an intelligent programming tutor. In E. Lemut, B. du Boulay, & G. Dettori (Eds.), *Cognitive models and intelligent environments for learning programming* (pp. 135–144). New York: Springer-Verlag.

Corbett, A. T., Anderson, J. R., & Patterson, E. G. (1990). Student modeling and tutoring flexibility in the LISP Intelligent Tutoring System. In C. Frasson & G. Gauthier (Eds.), *Intelligent tutoring systems: At the crossroads of artificial intelligence and education* (pp. 83–106). Norwood, NJ: Ablex.

Goldstein, I. P. (1982). The genetic graph: A representation for the evolution of procedural knowledge. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 51–77). New York: Academic Press.

Kieras, D. E., & Bovair, S. (1986). The acquisition of procedures from text: A production system analysis of transfer of training. *Journal of Memory and Language, 25*, 507–524.

Koedinger, K. R., & Anderson, J. R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. *Cognitive Science, 14*, 511–550.

Kulik, C. C., Kulik, J. A., & Bangert-Drowns, R. L. (1990). Effectiveness of mastery learning programs: A meta-analysis. *Review of Educational Research, 60*, 265–299.

Kulik, J. A., Kulik, C. C., & Cohen, P. A. (1979). A meta-analysis of outcome studies of Keller's Personalized System of Instruction. *American Psychologist, 34*, 307–318.

Larkin, J. H., McDermott, J., Simon, D. P., & Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science, 208*, 1335–1342.

Lesgold, A. M., Rubinson, H., Feltovich, P., Glaser, R., Klopfer, D., & Wang, Y. (1988). Expertise in a complex skill: Diagnosing X-ray pictures. In M. T. H. Chi, R. Glaser, & M. J. Farr (Eds.), *The nature of expertise* (pp. 311–342). Hillsdale, NJ: Lawrence Erlbaum Associates.

McCloskey, M. (1983). Naive theories of motion. In D. Gentner & A. Collins (Eds.), *Mental models.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Newell, A. (1990). *Unified theories of cognition.* Cambridge, MA: Harvard University Press.

Reed, S. K., Dempster, A., & Ettinger, M. (1985). Usefulness of analogous solutions for solving algebra word problems. *Journal of Experimental Psychology: Learning, Memory and Cognition, 11*, 106–125.

VanLehn, K. (1990). *Mind bugs: The origins of procedural misconceptions.* Cambridge, MA: MIT Press.