THE JOURNAL OF THE LEARNING SCIENCES, 4(2), 167–207

Cognitive Tutors: Lessons Learned

John R. Anderson, Albert T. Corbett,
Kenneth R. Koedinger, and Ray Pelletier
Carnegie Mellon University

This article reviews the 10-year history of tutor development based on the advanced computer tutoring theory (J. R. Anderson, 1983, 1993). We developed production system models in ACT of how students solved problems in LISP, geometry, and algebra. Computer tutors were developed around these cognitive models. Construction of these tutors was guided by a set of eight principles loosely based on the ACT theory. Early evaluations of these tutors usually, but not always, showed significant achievement gains. Best case evaluations showed that students could achieve at least the same level of proficiency as conventional instruction in one third of the time. Empirical studies showed that students were learning skills in production-rule units and that the best tutorial interaction style was one in which the tutor provides immediate feedback, consisting of short and directed error messages. The tutors appear to work better if they present themselves to students as nonhuman tools to assist learning rather than as emulations of human tutors. Students working with these tutors display transfer to other environments to the degree that they can map the tutor environment into the test environment. These experiences have coalesced into a new system for developing and deploying tutors. This system involves selecting a problem-solving interface, constructing a curriculum under the guidance of a domain expert, designing a cognitive model for solving problems in that environment, building instruction around the productions in that model, and deploying the tutor in the classroom. New tutors are being built in this system to achieve the National Council of Teachers of Mathematics (NCTM) standards for high-school mathematics in an urban setting.

Over the past 10 years, our research group (the Advanced Computer Tutoring Project at Carnegie Mellon University) has been developing a type of com-

Requests for reprints should be sent to John R. Anderson, Carnegie Mellon University, Department of Psychology, Pittsburgh, PA 15213–3890.

puter-based instructional technology that we call cognitive tutors. The core commitment at every stage of the work and in all applications is that instruction should be designed with reference to a cognitive model of the competence that the student is being asked to learn. This means that the system possesses a computational model capable of solving the problems that are given to students in the ways students are expected to solve the problems. As is elaborated later, all decisions about delivering such instruction are made with reference to that model. These systems are called tutors because our initial work on them was inspired by the intelligent tutoring work of the late 1970s and early 1980s (e.g., Sleeman & Brown, 1982). Indeed, when we embarked on the project, we had the ill-defined goal that our systems should interact with students like private human tutors. Although we deemphasized the emulation of the human tutors over the years, the term *tutor* has stuck.

This article surveys our work on tutoring. It describes the motivations for being involved in tutoring, the theoretical assumptions underpinning the work, the empirical evidence for the claims, and the current directions of the research. This overview is organized according to its three identifiable stages: (a) a flurry of tutor building in the mid-1980s, (b) a flurry of evaluations in the late 1980s, and (c) a current effort to build and deploy practical tutor systems.

STAGE 1: EARLY TUTOR BUILDING

The year 1982 saw the completion of the ACT* theory of learning and problem solving, which was described in the book *The Architecture of Cognition* (J. R. Anderson, 1983). Much of that theory was concerned with the acquisition of cognitive skills, and we tested the theory in the domains of proof generation in geometry (J. R. Anderson, Greeno, Kline, & Neves, 1981) and initial programming skills in LISP (J. R. Anderson, Farrell, & Sauers, 1984). The theory held that a cognitive skill consists in large part of units of goal-related knowledge. Cognitive skill acquisition involves the formulation of thousands of rules relating task goals and task states to actions and consequences. The theory employs a production-rule formalism to represent this goal-oriented knowledge. For example, a geometry proof generation rule might be:

IF the goal is to prove two triangles are congruent
THEN set as a subgoal to prove that corresponding parts are congruent.

A LISP programming rule would be:

IF the goal is to get the second element of the list
THEN code *car* and set as a subgoal to pass to car an argument that is the tail of the list.

It is not the production-rule notation that is critical; it is the set of representational features that this notation describes. For example, production rules are procedural, abstract, modular, directional, and goal-related. See J. R. Anderson (1993, chap. 2 and especially Section 2.4) for an elaboration of these features.

A theory of the acquisition of cognitive skills should have implications for their instruction. We thought it would be an important test of the theory if we could use it to optimize learning. However, it is not a trivial matter to convert a scientific theory of a phenomenon to an engineering theory of how to foster that phenomenon. We undertook our first work in intelligent tutoring to explore how such a conversion might take place.

The ACT Theory

Because the tutoring effort is so strongly tied to the ACT theories of skill acquisition (initially the ACT* theory; J. R. Anderson, 1983; and now the ACT–R theory; J. R. Anderson, 1993), it is worth reviewing the principal tenets of that theory.

Procedural-declarative distinction. The theory distinguishes between declarative knowledge (e.g., knowing the side-angle-side theorem) and procedural knowledge (e.g., an ability to use the side-angle-side theorem in a proof). The assumption of the theory is that goal-independent declarative knowledge initially enters the system in a form that can be encoded more or less directly from observation and instruction. Cognitive skill depends on converting this knowledge into production rules like those noted previously, which represent the procedural knowledge.

Knowledge compilation. It is assumed that the students could use various interpretive procedures, such as instruction following and analogy, to generate problem-solving behavior by relating declarative knowledge to task goals. A learning process called *knowledge compilation* converts this interpretive problem solving into production rules. Thus, the theory assumes that production rules can only be learned by employing declarative knowledge in the context of a problem-solving activity.

Strengthening. It is assumed that both declarative and procedural knowledge acquire strength with practice. Application of weak knowledge can result in slips and errors. Thus, even after the knowledge has been successfully encoded, further practice produces smoother, more rapid, and less errorful execution.

These three assumptions[1] pointed us in the direction of a method of instruction in which students were presented with an initial brief declarative instruction and then a good deal of guided practice. As stressed elsewhere (J. R. Anderson, 1993; J. R. Anderson, Conrad, & Corbett, 1989), this conception of skill acquisition is quite simple. The apparent complexity of learning a cognitive skill results from the inherent complexity of the domain being learned. That complexity is reflected in the complexity of the rule set that has to be learned, but the learning of each production rule is quite simple.

The Nature of a Cognitive Skill

We have already used the term *cognitive skill* and will continue throughout this article to use the term to describe what our tutors teach. Therefore, it seems important to be clear on how the term relates to what one might conceive of as "competence" in a particular domain, such as geometry. We use cognitive skill to refer to the set of production rules acquired in the domain. According to the ACT theory, there is more to domain competence than just these production rules. There are also the declarative structures that represent domain knowledge. Although, in principle, it is possible to have all domain knowledge represented in production rules or all domain knowledge represented declaratively (and being interpreted by domain-independent procedures), we do not think either is the profitable way to develop domain competence. If everything had to be represented as production rules, too many rules would be required because it would be necessary to represent each piece of knowledge in each way it could be used. If everything had to be interpreted from declarative representations it would be too inefficient and place too great a burden on working memory.

An example of a declarative structure in the domain of geometry might be the side-angle-side theorem: "If two sides and the included angles of two triangles are congruent, then the triangles are congruent." Procedural knowledge might involve skills of placing triangles into correspondence, determining what an included angle is, setting subgoals, and making inferences. It might also include some frequently encountered uses of this rule, such as recognizing triangles as congruent which meet this condition.

Given that competence depends on both declarative and procedural knowledge, why have we placed the emphasis on the procedural? This is because our view is that the acquisition of the declarative knowledge is relatively problem-free. However, declarative knowledge by itself is inert and often quite useless.[2] Declarative knowledge can be acquired by simply

---

[1]There were other ideas in the ACT* theory about production-rule learning, but these were abandoned in the ACT–R theory, which is in many ways a simplification of the earlier theory.

[2]Although our theoretical interpretation of the phenomenon is different, this issue of inert knowledge is, of course, a familiar problem in education (e.g., Brown, Collins, & Duguid, 1989; Cognition and Technology Group at Vanderbilt, 1990; Whitehead, 1929).

being told and our tutors always apply in a context where students receive such declarative instruction external to the tutors. What is problematical is acquiring the procedural knowledge that enables this inert knowledge to become the basis for effective action in the context of use. Production rules cannot be learned by simply being told. Rather, they are skills that are only acquired by doing. Thus, it is critical to set up contexts in which these skills can be displayed, monitored, and appropriate feedback given to shape their acquisition. This is the function of our tutors.

Initial Work on Tutoring

Our initial motivation in developing intelligent tutoring systems was mainly to learn more about skill acquisition rather than to produce practical classroom results. It was a significant test of the ACT theory to see whether we could produce successful learning by getting students to act like the underlying production-rule model. It was by no means obvious at the time whether or not there were going to be major gaps in ACT production-rule models when applied to such instructional situations.

In 1983 work began on a LISP tutor and a geometry tutor (J. R. Anderson, Boyle, Corbett, & Lewis, 1990; J. R. Anderson, Boyle, & Yost, 1986; J. R. Anderson & Reiser, 1985). The former helped students write short programs in LISP, and the latter helped students search for geometry proofs and represent them in proof-graph form. The screen displays for the two tutors are depicted in Figures 1 and 2. These two tutors embodied a number of key ideas about how computer-based instruction should be realized. These ideas have been part of all of our subsequent tutors.

Model. There should be a production-rule model of the underlying skill incorporated into the tutor. This is a model that would perform the task the student was expected to perform. At each point in the problem solving the model is capable of generating a set of production sequences that represent correct solutions of the problem.

On-path actions. Correct actions on the student's part are recognized if they are along one of the correct solution paths generated by the model. If the student is correct, the tutor does not comment but rather allows the student to progress with the solution.

Off-path actions. If the student performs an off-path action, instruction is focused on getting the student back on path. Our earlier tutors required students to always stay on path. More recent tutors allow the student to go off path but still focus instruction on getting the student back on path when they are off path.

Error feedback and help. The tutors possess two types of instruction. If the student makes a recognizable error (a "bug"), a message can be given explaining why it is an error. This is generated from a buggy production that embodies the error. If the student asks for help, a help message is presented to guide the student to the correct solution. This message is generated from the information along a correct path. Both bug messages and help messages are generated to be specific to the particular context in which they occur by using the particular instantiations of the general production rules.

This approach to tutoring is described as the *model-tracing approach* because it involves trying to relate the behavioral manifestations of the student's solution on the computer to some sequence of production firings in the cognitive model. This is a version of the plan-recognition problem, which is recognized as being computationally very difficult in its general form because of the combinatorics of how a plan can fit onto external behavior. We originally dealt with this problem by insisting that each action of the student be on an interpretable path. When there was any ambiguity about the interpretation of the student's action the student was presented with a disambiguation menu to identify the proper interpretation of the action. If the student's action was in error, the student was to correct it and

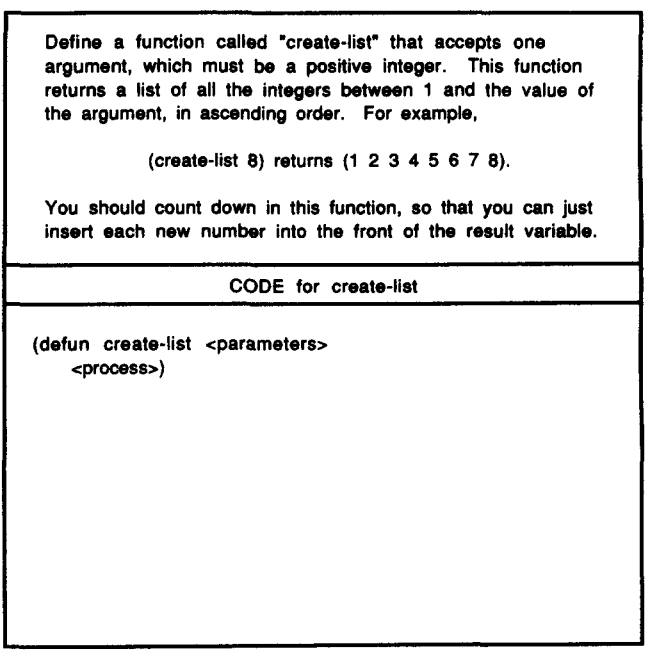

FIGURE 1 The appearance of the LISP tutor screen at the beginning of a coding problem. (From *Rules of the Mind,* p. 146, by J. R. Anderson, Ed., 1993, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc. Copyright 1993 by Lawrence Erlbaum Associates, Inc.)

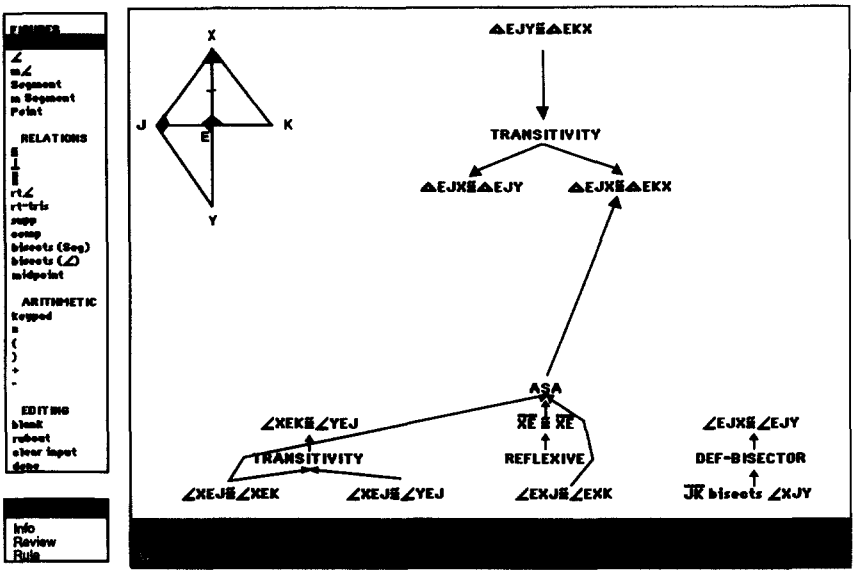

FIGURE 2 A screen image from the geometry tutor showing the proof-graph formalism. The givens of the problem are at the bottom of the screen, and the statement to be proven is at the top. (From *Rules of the Mind,* p. 171, by J. R. Anderson, Ed., 1993, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc. Copyright 1993 by Lawrence Erlbaum Associates, Inc.)

get back on an interpretable path. This approach, combined with an interface which yielded a relatively rich behavioral trace and a restriction on possible interpretations of the behavior, tamed the combinatorics of the problem so that we were able to follow the solution of the student and do so in real time. As we discuss later in this article, we have subsequently relaxed the requirement that the student stay on an interpretable path but have done so in ways that avoid the potential combinatorial explosion.

This technical accomplishment was no mean feat in itself. It was and still is the only practical automatic approach to protocol analysis. This model-tracing approach has been adapted to doing automatic protocol analyses of problem solving in psychology experiments where there is no tutorial intervention (J. R. Anderson, 1993). However, solving the technical problem of model tracing does not bring with it any automatic guarantee of instructional effectiveness.

Interacting With the LISP Tutor

We first need to illustrate what it is like to interact with one of these tutors. Because we report more empirical research from the LISP tutor than from the others, it is the best choice for an illustration. This section describes an interaction with the original LISP tutor we created.[3] Figure 1 depicts the

terminal screen at the beginning of an exercise. The original LISP tutor ran on Vaxes and communicated with students via a regular 24 × 80 character terminal. The screen is divided into two windows, and the problem description appears in the "tutor window" at the top of the screen. As the student types, the code appears in the "code window" at the bottom of the screen. This exercise is drawn from the lesson in which iteration is being introduced. Students are familiar with the structure of function definitions by this point, so the tutor has put up the template for a definition, filling in *defun* and the function name for the student. The symbols in angle brackets reify remaining goals, that is, they represent code components remaining for the student to supply. The tutor places the cursor over the first symbol the student needs to expand, <PARAMETERS>.

As the student works on an exercise, this tutor monitors the student's input, essentially on a symbol-by-symbol basis. As long as the student is on some reasonable solution path, the tutor remains in the background and the interface behaves like a structure editor. The tutor expands templates for function calls, provides balancing right parentheses, and advances the cursor over the remaining symbols that must be expanded. If the student makes a mistake, however, the tutor immediately provides feedback and gives the student another opportunity to type a correct symbol. If the student requests an explanation or if the student appears to be floundering,[4] the tutor will also provide a correct next step in a solution, along with an explanation.

Table 1 contains a record of a hypothetical student completing the code for the exercise.[5] This table does not attempt to show the terminal screen as it actually appears at each step in the exercise. Instead, it shows an abbreviated "teletype" version of the interaction. As described earlier, while the student is working, the problem description generally remains in the tutor window (except when a message to the student is being presented), and the code window is updated on a symbol-by-symbol basis. Instead of portraying each update to the code window in the interaction, the table portrays nine key "cycles" in which the tutor interrupts to communicate with the student. At each of these enumerated cycles the complete contents of the code window are shown along with the tutor's response. The tutor's response is shown below the code to capture the temporal sequence of events; on the terminal screen, the tutor's communications would appear in the tutor window above the code. In each cycle all the code that the student has typed since the

[3]As it turns out, there are a number of different LISP tutors that we have constructed. The original tutor is affectionately called "LISP tutor classic" in our laboratory.

[4]Students are judged to be floundering at a step in the solution when they repeat the same type of error three times or make two mistakes that the tutor does not recognize.

[5]Undoubtedly, the use of LISP creates a barrier to communication with that fraction of the readership that is not familiar with LISP. However, the semantics of LISP are not really necessary to understanding how the tutor interacts or how these interactions depend on the underlying production-rule models.

TABLE 1

A Teletype Depiction of a Hypothetical Student Completing the Code for the Function **create-list** With the Tutor

The Problem Description

Define a function called "create-list" that accepts one argument, which must be a positive integer. This function returns a list of all the integers between 1 and the value of the argument, in ascending order. For example:

(create-list 8) returns (1 2 3 4 5 6 7 8).

You should count down in this function, so that you can just insert each new number into the front of the result variable.

Cycles in which the tutor communicates with the student.

1. (defun create-list **(num)**
(loop)

You will need local variables in this function. So before you code anything else, there is a particular LISP operator you need to code.

---

2. (defun create-list (num)
(let (( )
<BODY>))

Which of the following do you want to do?
PRESS: IF YOU WANT TO
1 Initialize a variable to hold the list of digits between 1 and the value of the parameter.
2 Initialize a counter variable to keep track of how many times we go around the loop.
3 Have the tutor choose.

Menu choice: 2

3. (defun create-list (num)
(let **((count 1)**
<OTHER-INITIALIZATIONS>)
<BODY>))

In this problem we won't want to start counting at the same value each time we call the function, so we can't initialize the counter to a constant like 1. How is the starting value being communicated to the function?

---

4. (defun create-list (num)
(let ((count **num)**
(numlist (list num)))
(lop))

Did you mean loop (y or n)? **y**

---

5. (defun create-list (num)
(let ((count num)
(numlist (list num)))
(loop
(cond ((equal count 1) numlist))
<LOOP-ACTIONS2>)))

You do want to return the variable "numlist" here, but just typing it into the code won't accomplish that. What LISP operator will allow you to return something from a LOOP?

(continued)

TABLE 1 (*continued*)

The Problem Description

6. (defun create-list (num)
   (let ((count num)
         (numlist (list num)))
     (loop
       (cond ((equal count 1) **(return numlist)**))
       <LOOP-ACTIONS2>)))

In numeric iteration the first thing you should do after the exit test is update the counter variable. You can use the function SETQ to do this.

---

7. (defun create-list (num)
   (let ((count)
         (numlist (list num)))
     (loop
       (cond ((equal count 1) (return numlist)))
       (setq **count (1- count)**)
       (cons))))

You've got the right idea, since you'll need to use the function "cons," but remember you are trying to insert the new value of the counter into the list. What LISP function allows you to assign the results of this function call to a local variable?

---

8. (defun create-list (num)
   (let ((count num)
         (numlist (list num)))
     (loop
       (cond ((equal count 1) (return numlist)
       (setq count (1- count)
       (setq numlist (list))))

If you LIST together arguments like 7 and (8) you get (7 (8)). What you want is (7 8). Try another function that combines things together.

---

9. (defun create-list (num)
   (let ((count num)
         (numlist (list num)))
     (loop
       (cond ((equal count 1) (return numlist)))
       (setq count (1- count)
       (setq numlist **(cons count numlist)**)))))

–YOU ARE DONE. TYPE NEXT TO GO ON AFTER–
–TESTING THE FUNCTION YOU HAVE DEFINED–

THE LISP WINDOW

=> (create-list 10)

(1 2 3 4 5 6 7 8 9 10)

=> next

---

preceding cycle is shown in boldface in Table 1. However, in each case, the tutor is responding specifically to the last symbol the student typed.

In the first of the cycles displayed, the student has typed in the parameter list and has called *loop* so as to iterate. The tutor reminds the student that it is necessary to create some local variables before entering the loop. In the second cycle, the student has called *let* and is about to create a local variable. The template for numeric iteration calls for two local variables in this function, so the tutor puts up a menu to clarify which variable the student is going to declare first. This illustrates the tutor's need to know at all times what the student's intentions are so that it can follow the student. If there is an ambiguity, it will query the student by the means of such menus. In the third cycle, the student has coded an initial value that would be correct if the function were going to count up. However, this exercise is intended to give the student practice in counting down, so the tutor redirects the student. In the fourth cycle, the student has made a typing error that the tutor recognizes, and in the fifth cycle the student is attempting to return the correct value from the loop but has forgotten to call *return*. The tutor reminds the student that a special function call is required to exit a loop. The interactions between the tutor and student continue in this manner. Note that, for illustration's sake, this interaction shows students making more errors than they usually do. Typically, the error rate is about 15%, whereas it is approximately 30% in this dialogue. After each exercise, the student enters a standard LISP environment called the LISP window. Students can experiment in the LISP window as they choose; the only constraint is that they successfully call the function they have just defined (which the tutor has automatically loaded).

The Initial Incursion Into the Classroom

In 1984 we ran a few high-school students through the geometry tutor and taught a minicourse in computer science at Carnegie Mellon University (CMU) with the LISP tutor. The results exceeded our initial modest expectations. Students seemed to learn fairly well with the geometry tutor. The LISP minicourse was broken up into two groups to allow for an evaluation. One group worked exercises with the LISP tutor, and one worked the same exercises in a standard LISP environment. Students with the LISP tutor took 30% less time and scored one standard deviation higher on a final test than the students in the control condition.

In response to these results, we created a full semester course in LISP, taught to humanities and social sciences students, which is still a successful course today but with a number of revisions over the period.[6] We decided to

[6]Currently, the course is now a combined LISP and Prolog course in which the students learn both languages. The tutor is now delivered on a Macintosh system.

try to use the geometry tutor in a real classroom and were able to create a classroom of 10 Xerox D-machines, which were in Peabody High School in Pittsburgh from 1985 to 1988.

Emboldened by the prospects of practical tutors, we set out to create a third tutor for Algebra I; that tutor was also implemented on the D-machine. It tutored algebra symbol-manipulation skills (e.g., solving linear equations), which had been the focus of some discussion in cognitive science (e.g., Matz, 1982).

Figure 3 illustrates the appearance of the algebra tutor (for more details read Milson, Lewis, & Anderson, 1990). Our analysis of the algebra symbol-manipulation skills revealed that there were clear algorithms for accomplishing all tasks, and successful students had learned these algorithms. The algorithms were described in more or less detail in various textbooks. A key feature of these algorithms was that they were quite hierarchical, such that, to solve a linear equation, one might have to get rid of embedded expressions, which required distribution, which might require multiplication of fractions, which required integer multiplication, and so on. We observed that weaker students were frequently confused about a lower level operation, and it was difficult to get them to identify the level at which their problem occurred. To facilitate this remediation we attempted to illustrate this hierarchical structure by a representation like Figure 3, which placed boxes representing suboperations within boxes representing larger operations. This would allow us to identify and focus instruction on the level at which the student was having difficulties.

The Eight Principles

About that time it seemed that we should try to draw a tighter connection between the ACT* theory and our tutoring practice. J. R. Anderson, Boyle,

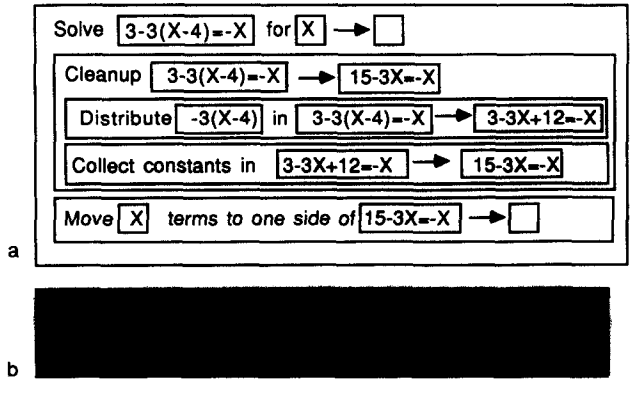

FIGURE 3 The box notation for representing the hierarchical algorithmic decomposition in the algebra tutor.

Farrell, and Reiser (1987) examined the ACT* theory and extracted what we felt were eight principles for design of tutors that followed from that theory. These principles follow.

Principle 1: Represent student competence as a production set. The fundamental insight is that the tutoring enterprise should be informed by an accurate model of the target skill. The cognitive model allows us to set appropriate curriculum objectives and to properly interpret the actions of the student. This has been the essential difference between our approach and the more behaviorist approaches to computer-based instruction. The production rules define a more abstract and, we believe, more accurate representation of the target skill than did the behavioral objectives of a typical behaviorist analysis (e.g., Bunderson & Faust, 1976; Gagné & Briggs, 1974). However, our approach shares with the behaviorist approach the idea of decomposing a skill into components and organizing instructions according to the componential analysis. The difference is in what the components are.

This principle does not specify how to define a computer interface, how to interact with a student, or when to promote the student through the curriculum. This all depends on a theory of how such production rules are acquired. The other principles of tutoring were all concerned with how to take this first insight and convert it into pedagogical policy. These other principles were derived with varying degrees of rigor from the ACT theory of skill acquisition.

Principle 2: Communicate the goal structure underlying the problem solving. One of the enduring assumptions of the ACT theory has been that solving a problem involves decomposing that problem into a set of goals and subgoals. Another observation was that, in many domains that students had difficulty mastering (e.g., proof skills in geometry or writing recursive programs), the goal structure governing the problem solving was not adequately communicated to the student. So the reasonable assumption was that exposing and communicating such goals should be an instructional objective. We adapted an approach that has been called *reification* (Brown, 1985; Collins & Brown, 1987). We attempted to develop interfaces that made explicit the goal structures that were only implicit in the instruction. We had at least two notable successes. This was the use of a proof graph in geometry to illustrate the subgoaling relationship between certain conclusions and the ultimate conclusion of the proof.[7] The other was Singley's use of a subgoal tree to illustrate use of the chain rule in related-rates calculus problems (Singley, 1986).

[7]For an evaluation of the contribution of the proof graph over and above any tutoring, see Scheines and Sieg (1993).

Principle 3: Provide instruction in the problem-solving context. This principle was based on the research showing the context specificity of learning (e.g., J. R. Anderson, 1990, chap. 7). The current situated learning movement (e.g., Collins, Brown, & Newman, 1989; Lave & Wenger, 1990) presumably gives a new currency to this principle. The difficulty with this principle is that there is not a detailed theoretical interpretation of why it is true, and so it is a little hard to know how to apply it in detail. Does this mean that instruction should be provided in the same class session as the tutor is used, before each problem, or in the midst of each problem? As it has evolved in our applications, this has come to mean providing instruction between each new section in the tutor (a section begins where new production rules are introduced), allowing the student to refer back to this instruction in the course of problem solving. We have experimented with placing instruction at the precise point where it is needed in a problem, but students find this interferes with their problem solving.

Principle 4: Promote an abstract understanding of the problem-solving knowledge. This principle was motivated by the observation that students will often develop overly specific knowledge from particular problem-solving examples. In terms of production rules, this has meant that the conditions on the rules were not sufficiently general. Although the problem is undoubtedly real, this principle provides no guidance for how it is to be achieved. In practice, we tried to reinforce the correct abstractions in the language of our help and error messages.

Principle 5: Minimize working memory load. This principle was motivated by the fact that learning a new production rule in ACT requires that all the relevant information (relevant to the condition and action of the to-be-learned production) be simultaneously active in memory. Keeping other information active could potentially interfere with learning the target information. Sweller (1988) showed that a high working-memory load interferes with learning. This principle means minimizing presentation and processing of information not relevant to the target productions. This includes minimizing presentation of instruction while problem solving because processing this instruction poses another working-memory load.

This also implies that one should try to provide instruction on specific components only when other components of the skill have already been relatively well mastered. This leads to a curriculum design in which only a few new things are taught at a time. This could be viewed as being at odds with the current approaches, such as cognitive apprenticeship or anchored instruction, which advocate teaching component skills in the context of complex, real-world problems. However, this approach does not deny the value of learning in such context but, rather, argues that students should

gradually acquire the skills required to deal with this complexity rather than having to acquire them all at once.

Principle 6: Provide immediate feedback on errors. This clearly has been the most controversial of our tutoring principles. The ACT* theory claimed that new productions were created from records of problem-solving traces. Therefore, the longer one waited until an error was corrected, the longer the span of problem solving over which the student would have to integrate to create a production. The current ACT–R theory claims that one learns from problem-solving products. Thus, the learner examines the resulting solution (code, proof, algebraic derivation) and builds productions from that. Thus, it does not matter whether all the critical steps occur together in time or not—only that they be represented in the final solution. Thus, the principal theoretical justification for immediate feedback no longer exists in ACT–R. We discuss later evidence about immediacy of feedback from our tutors that is consistent with the current ACT–R conception. Still, we note that immediate feedback can be beneficial in cutting down on time spent in error states and making it easier to interpret the student's problem solving.

Principle 7: Adjust the grain size of instruction with learning. This principle was motivated by the composition learning operator in ACT*, which claimed that single productions would be composed into larger productions that did in one cognitive step what had been done in many steps. Whereas ACT–R does not have such a composition learning operator, it still predicts this change in the grain size of problem solution, but from other mechanisms (J. R. Anderson, 1993, chap. 4). Thus, it seemed reasonable to design the interface so that one could process the student's problem solving in ever larger units of analysis. There has only been one early attempt to do this, however, and this was with the algebra tutor (J. R. Anderson et al., 1990). That attempt was not notably successful. In retrospect, our problems here reflected some fundamental misconceptions about the role of the interface in problem solving. This is a topic that is discussed at length later in the article.

Principle 8: Facilitate successive approximations to the target skill. Frequently, when students are initially trying to perform a skill, they cannot perform all the steps. We had the tutor fill in the missing steps. The expectation was that with repeated practice this division of labor between student and tutor would change, with the student providing more and more of the work until the tutor was completely in the background. In practice, this successive approximation has frequently worked quite well. This principle seems quite analogous to "fading" in the cognitive apprenticeship terminology (Collins, Brown, & Newman, 1989).

Some of these principles are similar to ideas that accompanied more behaviorist attempts at instructional design (Bunderson & Faust, 1976; Gagné & Briggs, 1974). This is particularly true for Principles 3, 6, 7, and 8. The difference is that these principles were being used in service of a different representation of the underlying skill. The places where these principles add something to the standard behaviorist approach (Principles 1, 2, 4, and 5) reflect the different representational assumptions. This is a case where assumptions about knowledge representation matter.

The fact that our tutors embody cognitive models of the target competence does not imply that they would always behave differently from instructional systems based on behaviorist principles: It depends on the domain. If we were building a spelling tutor with the goal of memorization, we suspect it would be much like behaviorist applications (e.g., Porter, 1961), which produce similar achievement gains to those of our systems. However, we have chosen to focus our applications on much more complex skills, in which our cognitive models do lead to different instructional strategies. It is our impression that the behaviorist programs have not had much success in extending to such complex domains.

STAGE 2: THE EVALUATIONS AND EMPIRICAL STUDIES

J. R. Anderson et al. (1990) reported the state of the tutoring work in 1987, including the results of the first phase of research activity. This section reviews those results and brings the research record up to date. The first three sections describe summative evaluations of the geometry tutor, algebra tutor, and LISP tutor. Succeeding sections discuss evidence on the componential nature of skill acquisition, student modeling, and feedback control and content.

The Geometry Tutor

The geometry tutor was used in a pilot study in the 1985–1986 school year. A number of classes were exposed to it, and all showed large achievement gains. The 1986–1987 school year was the major test in which we compared classes with the tutor with classes without the tutor but with the same teacher. We performed a number of regression analyses trying to predict student performance in a final paper-and-pencil test of proof skills. The following equation predicted student performance on a scale from 0 to 80:

$$
\begin{aligned}
&35 + 7.5^{*} \text{ (letter grade in algebra)} \\
&\quad + 14 \text{ if access to tutor one on one} \\
&\quad + 4 \text{ if access to tutor two on one}
\end{aligned}
$$

The student's letter grade in the prior year's algebra class (1 = D to 4 = A) was the best measure of prior individual student differences in predicting geometry test performance (e.g., better than IQ). The 14 points for the tutor reflected more than one standard deviation in the population or more than one letter grade on the test. Because we did not have enough machines, sometimes pairs of students worked on the machines. In this case, most of the tutor benefit was eliminated, and the remaining 4-point advantage of these students was not statistically different from that of the control group.

In addition to our own assessment of the tutor, there have appeared reports from third-party observers (Schofield & Evan-Rhodes, 1989) and the teacher (Wertheimer, 1990) confirming the large positive impact of the tutor on the classroom. Schofield and Evan-Rhodes reported large improvements in the motivation of students, with students spending more time on task.[8] Wertheimer (1990) reported that he found the experience satisfying as a teacher because it allowed him to focus on the specific difficulties of specific students.

A more recent geometry tutor has been completed based on the cognitive model of geometry proof of Koedinger and Anderson (1990). It has been subject to a preliminary evaluation (Koedinger & Anderson, 1993) in which we also found a large positive result but only for the teacher carefully integrated into the project. Students with the tutor and the project teacher averaged just over five proofs correct out of eight, whereas students in each of the other three conditions (project teacher without the tutor, tutor without nonproject teacher, or nonproject teacher without tutor) averaged just over three proofs correct.[9] The fact that the tutor had its benefit only for the project teacher highlights the issue of integrating the tutor into the classroom.[10]

The Algebra Tutor

An evaluation of the algebra tutor was performed in the 1987–1988 school year. There were no differences between experimental classes, which had access to the tutors, and control classes, which did not. We think the major reason for the lack of effect was that there was a large difference between the tutor interface and the interface used in class (i.e., paper and pencil). It was just not obvious how to map the boxed representation of algorithmic decompositions (see Figure 3) to the linear line-by-line transformations that were used to assess performance in the paper-and-pencil posttest. A less important reason relates to a ninth-grade algebra class in an urban school. Symbol

[8]It got to the point where fights were occurring among students for access to the tutor.

[9]This is one standard deviation difference.

[10]Subsequent research has suggested that teachers take about 1 year to become comfortable with the tutor, and 2nd-year teachers have students who show achievement gains with the tutor.

manipulation is sufficiently easy that some students were mastering the skill quite well without tutorial intervention. Other students were just not involved in the class at all (often not attending), and the algebra tutor was too peripheral a part of their experience to help change their general pattern of behavior toward school.

We followed up the algebra tutor with a word algebra tutor that had some large positive results in the laboratory (Singley, Anderson, Gevins, & Hoffman, 1989), although it was never tested in the classroom. We think there are two basic reasons for the success of the word-problem tutor in the laboratory. First, the mapping from word problems in the tutor to the paper-and-pencil posttest was obvious. Second, word problems are in fact a difficult topic for even motivated students, and we were able to illuminate their instruction with our cognitive model for their solution. We are currently working with yet a newer algebra word tutor that is being used in the Pittsburgh Public Schools. Preliminary evaluations again suggest significant achievement gains.

The LISP Tutor

The LISP tutor was evaluated in a classroom setting early in its development. In the fall of 1984 we taught a minicourse on LISP in which students attended lectures and completed a fixed set of programming exercises either with the tutor or in a standard LISP environment. Students using the tutor completed the exercises 30% faster and performed 43% better on a posttest. We performed a second laboratory evaluation that more closely approximates the current self-paced course structure (Corbett & Anderson, 1991). Students worked through the lessons, reading the same text and trying to solve the same set of exercises with and without the tutor. In this evaluation, students using the tutor completed the exercises 64% faster and scored 30% higher on posttests than did students using a standard LISP environment. As is elaborated later, we think that the only reason for the posttest difference is that students working in the standard nontutor environment were unable to generate working solutions to all the exercises. Conceivably, if students in the control condition had put in sufficient time, they could have eventually found working solutions and scored as well on the posttest. Nonetheless, this study supports the claim that a well-designed tutor can bring students to as high or higher achievement levels in no more than one third the time required by traditional learning environments.

A more typical practice in education evaluation is to hold learning time constant and examine differences in achievement scores. This is what we were forced to do in our algebra and geometry evaluations because we could not manipulate the time students spent on these problems. The achievement differences are always a little hard to judge—what does 14 points on an 80-point test really mean? One solution is to report differences in achievement level measured in standard deviations (e.g., Bloom, 1984). However,

standard deviation differences say as much about the variance in test scores as they do about the impact of an instructional manipulation. Such variances are substantially affected by test construction and inherent variability in the population. Thus, the numbers are virtually meaningless except to establish the direction of the difference. It is more meaningful to hold constant the level of mastery required and look at differences in time to achieve that level. This reflects the true gain of an educational technique.

The LISP tutor has been followed up with a general programming environment in which LISP, Prolog, and Pascal tutors have been built (J. R. Anderson et al., 1993). The Pascal tutor has just started to be used in the public schools. This will serve as a basis for evaluation outside of the rather specialized CMU population.

Componential Analysis of Learning

One of the current controversies in cognitive science and education is whether it is possible to take a complex competence, break it down into its components, and understand the learning and performance of that competence in terms of the learning and performance of the components (e.g., Shephard, 1992). When we have addressed this question in the context of our tutors, the answer is a resounding yes.

Consider the question of predicting how quickly and accurately a specific student will generate a particular piece of code in a particular LISP program. In a rather exhaustive analysis of data from the LISP tutor, J. R. Anderson et al. (1989) concluded that there were essentially four critical factors.

Production practice. The first factor was how often the student had applied the relevant production rule earlier. As students have more opportunities to use a production rule across exercises, their performance on the rule improves. Because there is a many-to-one mapping between production rules and surface code symbols (e.g., car, +, write, for) in different contexts, we are able to show it is the rule and not the surface construct that is the critical unit of practice. Similar production-specific learning has been shown in the case of geometry in which there is a many-to-one mapping between production rules and surface rules like "side-angle-side" (J. R. Anderson, Bellezza, & Boyle, 1993). In the ACT theory, this is attributed to strengthening the production rule. Figure 4 shows learning curves for "new" productions being introduced in a LISP lesson and "old" productions introduced in previous lessons. As can be seen, both show improvement as a function of amount of practice within the lesson. Old productions are better off because of practice from previous lessons. J. R. Anderson et al. (1989) also showed that student performance on old productions in a new lesson starts off close to where it left off on the previous lesson with only a little forgetting.

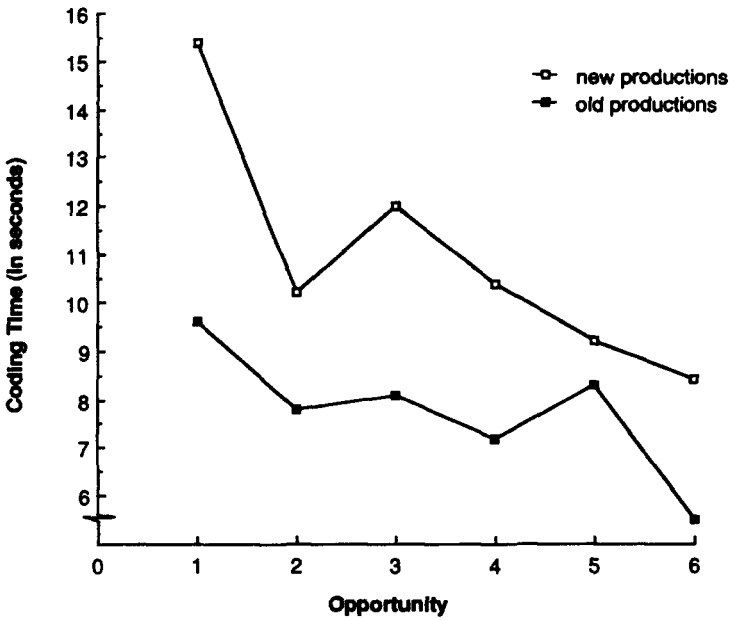

FIGURE 4 Time to write a piece of code in the LISP tutor for productions new to a lesson and old productions. Similar functions are obtained for error rates. (From *Rules of the Mind,* p. 153, by J. R. Anderson, Ed., 1993, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc. Copyright 1993 by Lawrence Erlbaum Associates, Inc.)

Within-problem practice effects. In both LISP and geometry, we were able to show that time and accuracy for rule application improves as the student progresses further into a specific problem, partialing out any effect of rule-specific practice. In the ACT theory, this is attributed to strengthening of the declarative representation of the problem through repeated access.

Acquisition factor. In a factor analysis of student performance, we found that students varied in how well they performed on new rules that were introduced in a lesson. The real significance of this factor is unclear. It may reflect some profound individual differences or just the care with which students reviewed the material.

Retention factor. The same factor analysis identified students who did well in retaining productions from earlier lessons. This factor was largely orthogonal to the acquisition factor. Again, the real significance of this factor is uncertain. It could reflect some profound individual differences or just how much students reviewed material between lessons.

The upshot of this analysis is the following scheme for predicting how well a student will do on a fragment of code: First determine if an old or new

production is generating that code. If it is a new production, one needs to use the learning curve for new productions to figure out the within-lesson practice effects, add in a factor to represent how much the student has worked on that problem, and add in an individual difference effect to reflect where that student stands on the acquisition factor. To predict performance on an old production one adds in the within-lesson practice effect for old productions, the problem practice effect, and an effect to reflect where that student is on the retention factor. As far as we could determine, these considerations captured the predictable variance.

Knowledge Tracing

The LISP tutor had a student modeling facility called *knowledge tracing* shared by neither the geometry nor algebra tutors.[11] As a student worked through the exercises, the tutor used a Bayesian procedure to estimate the probability that the student had learned each of the rules in the cognitive model. Knowledge tracing was used to implement a form of mastery learning. Students were given sufficient practice in each section of the curriculum to bring them to a specified degree of mastery of the individual cognitive rules introduced in the section before proceeding to the next section. This feature has substantial impact on student achievement level (J. R. Anderson et al., 1989). Knowledge tracing is a regular feature of all the tutors we are currently developing (see the section on practical deployment in this article).

A more detailed examination of the knowledge tracing model provided further confirmation of the componential analysis in the cognitive model: The learning and performance models that underlie knowledge tracing in the tutor can be used to predict posttest performance. The probability that a student will solve each posttest exercise correctly can be accurately derived from the probability that the student has learned each of the necessary rules (Corbett & Anderson, 1992; Corbett, Anderson, & O'Brien, 1995).

These results have strong implications for instruction. They imply that we should be able to get students to master the overall skill by getting them to master the individual components. Numerous analyses have reported positive results for mastery-based curricula (e.g., Block, 1971: Guskey & Gates, 1986; Kulik, Kulik, & Bangert-Downs, 1986), although the interpretation of these results is not without controversy (e.g., L. W. Anderson & Burns, 1987; Guskey, 1987; Slavin, 1987). Our application of mastery principles is different from most other efforts in that it is done on an individual student basis and it applies to the detailed components of the target skill.

---

[11] Public school teachers have been unwilling to allow students to progress at their own rate as enabled by the knowledge-tracing facility. Recently, we have gotten Pittsburgh Public School teachers to accept such individual progress.

Locus of Feedback Control

As prescribed by one of the original tutoring principles, our tutors conventionally employ immediate feedback and require immediate error correction. The LISP tutor has served as the vehicle for some studies that differentially distribute the control of feedback and the timing of error correction between the tutor and student. We developed three new versions that vary widely on the dimension of student/tutor control. At the far extreme from immediate feedback and control, we created a version that provides no advice on how to achieve programming goals. Students enter their code with a structure editor and have access to a LISP interpreter, but are largely on their own. This tutor provides just one bit of information: At any time in the course of problem solving students can ask whether their solution is correct (similar to checking an answer at the back of the book). This provides the best control against which to measure the effectiveness of our tutors, because it holds constant the type of problem-solving interface, the nontutor instruction, and the exercises attempted.

The remaining two versions are capable of providing the same advice as the standard immediate feedback version, but do so under different circumstances. One version, which we call the *error-flagging tutor,* falls closer to standard immediate feedback. This tutor identifies an error as soon as it occurs, by flagging it in bold font on the screen, but provides no feedback message and does not require immediate correction. The student can ask for a feedback message (the same one that would be presented automatically in immediate feedback), try to fix the error without feedback, or continue generating new code and come back to the error later. We call the other version the *demand tutor* because it provides no assistance until asked by the student. This tutor appears like the no-feedback control version as the student works, unless the student asks for error feedback. At that point, the tutor will identify the first error in the code (if any) and provide the same message as appears automatically in the immediate feedback version. In both of these versions, the student can ask for advice on how to accomplish a programming goal in addition to error feedback. The same advice is available at each goal in these tutors as in the standard tutor.

In the no-feedback, demand-feedback, and error-flagging tutors, the student might generate a complete working solution that the tutor does not recognize (cannot generate). As a result, the tutor will try any code that it does not recognize on a set of test cases and will accept the solution if it works. In practice, this happens rarely. Only about 5% of the students' unrecognized solutions worked.

We compared the four versions of the tutor across a five-lesson sequence that took students from the easiest (introductory) lessons to the most challenging (recursion) lessons (Corbett & Anderson, 1991). Each student attempted a fixed set of exercises (not necessarily enough to reach mastery) with one of the four versions. Students completed a paper-and-pencil posttest

and an online posttest in a standard LISP environment. There were no significant differences among the immediate-feedback tutor, flag tutor, or demand-feedback tutor in either of these posttest environments. Mean scores across the two posttest environments were 55%, 55%, and 58% correct, respectively. All three groups were reliably superior to the no-feedback control group (43% correct) in both posttest environments.

The time to complete the tutor exercises is displayed in Figure 5. As can be seen, the conditions are ordered in terms of tutor support: Immediate feedback is best, and it is followed by error flagging, demand feedback, and no-feedback.[12] Students in the three feedback conditions necessarily arrived at working solutions to each exercise. Whereas students in the no-feedback control condition attempted every exercise, they failed to solve 25%. Thus, the only condition to show inferior posttest performance was the condition in which students failed to solve all the problems. This reinforces our conclusions (J. R. Anderson et al., 1989) that posttest performance is primarily governed by the set of exercises that students successfully solve and understand.

Figure 5 depicts the 3:1 elapsed time ratio cited earlier between immediate feedback and no-feedback. This ratio underestimates the true benefit because students did not solve all the exercises in the no-feedback condition. It may underestimate the benefit over a typical classroom in that students in all conditions at least had access to declarative instruction and problems carefully designed to communicate and teach the target productions. A typical classroom may well be less organized.

An analysis of students' performance in the error flagging and demand feedback conditions indicated that students in these two conditions responded fairly passively to the control they were offered. When errors were noted immediately in the error flagging condition, students fixed the errors immediately 80% of the time. On the other hand, when the tutor did not volunteer error information in the demand-feedback condition, students rarely interrupted their coding activity to ask for help or evaluation. In the demand-feedback conditions, students did not ask for feedback until they had completed a preliminary solution in 90% of the exercises. Finally, we asked students how well they liked the tutor and the feedback they received. Perhaps surprisingly there were no reliable differences across groups, although there was an interaction with the curriculum: The harder the exercises became, the more students appreciated immediate help.

Feedback Content

A key feature of a tutor is what it says to the student during a problem-solving episode. There are two obvious occasions for communicating to the student during problem solving. One is when the student makes some error

[12]Subjects are taking longer in the later lessons because the exercises are longer and harder.

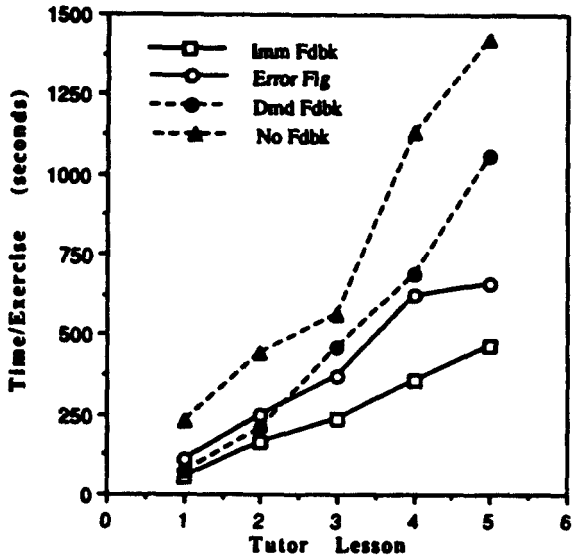

FIGURE 5 Mean exercise completion time for five tutor lessons. (From *Rules of the Mind*, p. 160, by J. R. Anderson, Ed., 1993, Hillsdale, NJ: Lawrence Erlbaum Associates, Inc. Copyright 1993 by Lawrence Erlbaum Associates, Inc.)

and the tutor can comment on the error. The other is when the student asks for help or appears to need help and some help message is given. Obviously, if students never get any information about errors in their solutions, they are not going to learn to avoid them. Similarly, if students never receive any help of any sort, they are in danger of becoming permanently stuck on some problems. However, one can construct a tutor in which errors are just flagged as such and correct solutions pointed out without any accompanying explanation. One can then ask what the potential benefit is of the accompanying explanations. We have performed such comparisons twice—once with the LISP tutor and once with the geometry tutor.

In the LISP tutor (J. R. Anderson et al., 1989), we took a number of measures of how much error messages helped. One was how well students performed on individual productions. We found students made fewer errors per production if they were receiving explanatory feedback (15% vs. 22%). Also, when they made an error and received feedback (an explanation, not just that they were wrong), they were more likely to correct their error on the first attempt (65% corrections vs. 38%). However, when we looked for long-term learning benefits we failed to find any significant differences. On a quiz immediately after the tutor exercises, students with explanatory feedback got 90% correct and those without got 91% correct. When we looked at their performance on a final exam, students with explanatory feedback got 76% correct and those without got 80% correct. Neither difference approached statistical significance.

Thus, the impact of such instruction in the LISP tutor was to facilitate the students' progress through the material but did not have any permanent achievement consequences. This is not an insignificant outcome because speed of learning is a critical dependent measure. Reasonable feedback messages also appeared to have a positive impact on the perception of the tutor. That is, students had numerous derogatory comments to make about the no-feedback tutor even if they eventually learned as much with it.

McKendree's (1990) evaluation of the feedback messages in the geometry tutor came to somewhat different conclusions. In her first dissertation study (McKendree, 1986), she found, as in the LISP tutor, that these messages facilitated progress through the tutor but did not have any permanent benefit. Frustrated with this lack of permanent benefit, she went through and specifically tuned the feedback messages to be particularly cogent to the specific problems. In her second evaluation (reported in McKendree, 1990), she was able to show a benefit in terms of both progress through the tutor and final achievement.

McKendree (1990) performed a theoretical analysis of why her students benefited from the carefully crafted feedback messages. She was able to show that students had failings in their underlying declarative knowledge that the feedback was able to correct. Some students without the feedback were able to get through the tutor without really correcting their misunderstandings and the holes in their knowledge. The tutor had no mastery-based instructional curriculum, and students just had to get through a fixed number of problems. This suggests that we have a time-achievement tradeoff and so suggests a way of reconciling the results with the LISP tutor. In the LISP tutor experiment, students were given enough problems to reach a high level of achievement, and the effect of feedback was on their time to reach that level. In the geometry tutor, students went through the problems in relatively constant time (and a short amount of time in hours), and the effect showed up in their achievement levels.

When one designs help messages, one tends to wax on in the messages to the student both to make the tutor seem intelligent and to communicate one's insights into the problem. Students take a rather different attitude. They realize it is just a computer that, at best, is just a tool to help them learn, and they have no interest in someone else's prose. They want to solve the problem and are often impatient with long messages. In a study with the algebra tutor, Lewis (1989) compared terse messages with longer messages more like natural English, which were originally used with the tutor. He found that students actually did better with the shorter messages, although the effect was not statistically significant.

STAGE 3: PRACTICAL DEPLOYMENT

Despite the successful empirical evaluations of our work on tutoring, our tutors had not been used much. The programming tutor has been regularly

used at CMU and occasionally used elsewhere. A scaled-down version of the geometry tutor was ported to a Mac SE, and that had been used occasionally in classrooms around the country. However, until very recently, our software had not played a significant and permanent role in the instructional plans of any organization outside of our own CMU classrooms. As long as the machines we were working with were large, impractical, artificial intelligence behemoths, this lack of practical demonstration was not a salient issue. However, now our tutors can be deployed on machines that are conceivable in American classrooms.

When we examined why we were so far from practical tutors it became apparent that we had avoided addressing a number of issues:

1. There was never any attempt on our part to address the curriculum that educators wanted to teach. The most striking case of this is the situation with our geometry tutors, which focused on teaching proof skills while mathematics educators were stressing more general reasoning and problem-solving skills.
2. There was no thought given to what would happen to students after they passed through our tutors. We always took as our measure of success performance on some final test. However, to be educationally valuable, our tutors have to fit in with some larger set of curriculum objectives.
3. The systems that we developed were inflexible in the way they had to be used and gave teachers no ability to tune the application of the tutors to their own needs and beliefs about instruction.
4. There was little understanding of how to support the deployment of these tutors in the classroom. Relevant to this is the observation that we have not had a positive classroom evaluation that did not involve teachers who had spent extensive time involved in the project.[13]

Addressing these problems has caused us to go beyond our original goals of showing that our cognitive models can lead to successful learning. We have now begun to address the issues of how to develop tutors that will implement an externally specified curriculum, which can be deployed in a wide range of classrooms, and which will leave students with a competence that makes a demonstrable contribution to their activities outside of the specific domains taught by the tutor. We have undertaken two major endeav-

[13]This, of course, raises interesting issues about evaluation. Because we have not gotten positive results simply by putting computers in the classroom, this indicates our positive results with project teachers is not simply a Hawthorne effect. The control classes (no-tutor classes) taught by our project teachers do at least as well as the control and tutored classes of the nonproject teachers. Thus, there is something special about the combination of the tutor and the teachers' preparation to use them. We are working with the Pittsburgh Public Schools to try to develop an appropriate teacher training program.

ors in response to these new agenda. One has been to create a development system for creating such cognitive tutors and to begin to work on a development discipline for their creation. The second has been to strike up a close relationship with the Pittsburgh Public Schools in which they serve as our clients and we try to build instructional software that can be used in their classrooms.

We have three permanent classrooms of Mac IIs and Quadras in three local city high schools, and additional classrooms are currently being planned in other area high schools. We are working toward supporting the city's mathematics and computer science curriculum. The emphasis in the mathematics curriculum is to implement the National Council of Teachers of Mathematics (NCTM) standards (NCTM, 1989) in an urban setting. A major issue here is to teach a curriculum that will empower students to participate in modern society. This is a particularly significant issue in a large urban school system with many students coming from economically disadvantaged families.

We have created a succession of development systems (J. R. Anderson, Corbett, Fincham, Hoffman, & Pelletier, 1992; J. R. Anderson & Pelletier, 1991). We have implemented in them tutors for three programming languages (LISP, Pascal, and Prolog), for elementary arithmetic, and for Algebra I. We have plans to build a tutor for geometry in this system that will extend the geometry tutor of Koedinger and Anderson (1993) to combine construction, exploration, conjecture, and proof. A major goal in this tutor development system is usability. This means both facilitating the teachers' use and modification of the systems and enabling as many people as possible to develop software in the system.

The actual process of developing a tutor has five identifiable stages: interface construction, curriculum specification, cognitive modeling, design of instruction, and classroom deployment. We discuss each of these.

Interface Construction and the Issue of Transfer

The first step in developing a tutor is to define the world in which the student's problem solving is going to take place. This will be the interface between the student and the computer. Placing interface design ahead of production-system design represents a major restructuring of our approach to tutor construction. In our early efforts, we started with an abstract production-rule model of the cognitive skill. Interface design was a secondary though nontrivial task in which we considered optimal ways to communicate the content of productions and to depict goals on the screen. Our current view is that the skill we are teaching is problem solving in a particular interface. Therefore, the interface must be designed before we can identify the production rules. The significant issue that we must face in interface design is transfer. What interface students learn will have a large impact on where their skills will transfer.

In designing an interface, one must keep in mind the domain to which the skill is supposed to transfer. Often that domain is still paper and pencil. For

instance, most college mathematics departments still expect incoming students to be proficient at paper-and-pencil algebraic manipulations. However, there is an increasing tendency for the target skill to involve use of computer software. Thus, part of the competence we are trying to teach in the current algebra tutor is how to use spreadsheets, symbol manipulation packages, and graphing routines. Having identified the target skills, one must design the interface to enable transfer to these target skills. The issue of transfer is one of psychology, and here it is worth distinguishing three levels at which transfer can occur.

Identical productions. The production rules for tutor exercises may be identical to those for the target domain. In this case, we would expect total transfer. This might be the case, for instance, if the target domain were a computer system and our tutor taught how to use it. In other cases, the tutor productions will only overlap with those for the target domain. In this case, Polson and Kieras (1985) and Singley and Anderson (1989) showed that the amount of transfer will be a function of the degree of production-rule overlap. For instance, our programming tutors focus on code selection and not syntax with a structure editor providing the syntax. Thus, students graduating from our tutors are successful coders but have some difficulty with syntax when tested without the structure editor because they have not acquired the necessary syntax productions. Although students have to pick up syntax after mastering the other aspects of the language, this does not appear to be a major learning hurdle (Goldenson, 1989a, 1989b).

Translating actions. Even if the tutor productions and target domain productions are different, it may be apparent to the student how to convert the actions of the learned productions into appropriate behavior in the target domain. For instance, we find relatively high transfer from a tutor that has students select programming constructs by menu to a test environment that requires the student to recall these constructs. This is because it is pretty apparent to most students that if they have been selecting "writeln" from a menu in the tutor, they should type that in when writing code into a standard file. In the 1986–1987 geometry tutor study, we found high transfer of students from doing proofs in the proof-graph formalism to the two-column proof formalism. This is a less obvious translation, but the teacher had gone over with them how the proof graph related to the traditional two-column proof.

Declarative transfer. Even if actions in one domain cannot be directly translated to actions in another, there can be declarative transfer of the underlying competence. Thus, we find that students who practice coding with the LISP tutor do fairly well in evaluating LISP expressions (although

there is hardly total transfer between these activities; J. R. Anderson et al., 1989). This is because both skills rest on the same declarative understanding of LISP and students must get their declarative representation right before they can acquire successful productions for coding.

There has been a great deal of interest of late in occasions where students fail to transfer across domains (Lave & Wenger, 1990). In our perspective, there is nothing mysterious about when transfer will occur and when it will not. Transfer requires students to have learned production rules in the training domain that will solve problems in the target domain. In the cases that the actions of the rules in the training domain are different than what is needed in the target domain (e.g., menu selection vs. writing), it must be apparent to the student how to map one action into another. In some writings, one gets the impression that lack of transfer is the rule. However, our research on tutoring shows that transfer as predicted by production-rule overlap is quite common. Every time we report a positive result in a paper-and-pencil test outside of one of our tutors it is a case of transfer.

There are two approaches to interface construction in our tutor development package. One is to build the interface ourselves. We have a set of primitives for facilitating the development of such interfaces and relating these interfaces to our production model. The other possibility is to take the existing piece of software and add hooks to it so that it is linked into our tutoring system. In either case, the following are the requirements for the interaction with the interface:

1. Actions taken to the interface must be passed through the tutor. The tutor needs to know what actions students have taken so it can follow students along the solution path they are pursuing and provide appropriate guidance.
2. The tutor must be informed about the consequences of any interface action for the state of the interface. Basically, the cognitive model needs to maintain in its working memory a representation of the interface that the students see.
3. The tutor must be able to perform interface actions itself.

Subject to these constraints, there is unlimited flexibility in the kind of interfaces we can tutor. We are particularly attracted to taking generally available pieces of software and tutoring students on problem solving within that software. When students leave our tutors, they will still have a useful problem-solving environment (and the transfer problem is minimized). So, for instance, we are doing some work on tutoring students on algebraic problem solving using Excel and geometric conjecture using Sketchpad (Key Curriculum Press, 1991). In using such an interface, we can take advantage of the many years of effort that went into making it flexible, reliable, and efficient.

Curriculum Construction

Once having specified the environment in which the students are going to display their problem-solving competence, we come to the issue of exactly what competence they are to display in that domain. Specifying the competence comes down to identifying the type of problems students are expected to solve in the domain and the constraints on their problem solving. Here our attitude is to take our specification from the educational community that is our client. For instance, in working with the Pittsburgh Public Schools mathematics educators, we take their input as to what problems they want students to solve. Fortunately, we have chosen a client who is at the lead in trying to achieve NCTM standards in an urban environment. Therefore, we are confident our tutors will have broader applicability.

Our clients also have strong input on the computer problem-solving interface. So, for instance, in the case of geometry it was the Pittsburgh Public Schools that decided we should work with the Geometer's Sketchpad (Key Curriculum Press, 1991). However, the exact form of that problem-solving interface was already determined by other forces. They made their choice much as they would make a textbook adoption. They cannot specify the microstructure of the interface any more than they can specify the exact content of a textbook. However, like a textbook, they can specify how the interface is to be used.

Within a particular interface, our clients have their conception of the problems they want students to solve and the constraints under which they want students to solve the problems. The issue of constraints is key here. For instance, a client may want to use a piece of software that has an algebraic-symbol manipulation package but may want to prevent the student from using that package in certain parts of the curriculum to exercise that student's own symbol-manipulation abilities. This amounts to the sort of instructions a teacher might give the student about how to solve a problem.

Although our software is initially developed in response to the needs of one client, other clients may want to use it with somewhat other goals in mind. This means we must allow them to select the constraints under which the problems are solved and the problems that are actually solved. It is useful to have a facility so teachers can enter new problems. Also, educators need to have access to some of the tutoring options. Earlier we described the variety of tutoring modes, such as immediate feedback, flag tutoring, and demand tutoring. Our current tutor development kit permits all of these different tutoring modalities, and the educator is able to choose among them.

Our tutors will track the students' performance on various production rules (knowledge tracing) and promote students through the curriculum as they achieve mastery on these rules (mastery learning). Again, teachers need to have the ability to turn mastery learning or knowledge tracing off or to override these facilities at various points.

Production System Modeling

Specifying a problem-solving environment and a set of constrained problems to be solved in that environment amounts to a behavioral specification of the target competence. Our major task as cognitive modelers is to figure out what that competence means in terms of a set of underlying production rules that are capable of generating that behavioral competence in a cognitively plausible way. This is the task of constructing a student model. Such a student model is able to be utilized in the sense that it can send actions to the interface that would constitute a correct solution to the problem. In most cases, there is more than one possible solution path, and the ideal student model must be capable of nondeterministically generating all the solutions.

The production rules respond to information in working memory. Typically, information in working memory will be of two kinds: information about what the current state of the problem is and a representation of what the goal is. In some cases, like the statement to be proved in a geometry problem, the goal representation is straightforward. However, when the goal is stated in natural language, as in the case of a programming problem, it can be quite problematical how to represent it. We do not want to represent or model the natural language processing that is involved in understanding the statement: This would just be too much to feasibly model. Therefore, we represent, in some form, what we believe to be the product of the natural language understanding. The problem is that it is hard to resist building into that representation part of the solution to the problem. Thus, we may not adequately represent the problem that the student faces or the skills that need to be learned.

Once the production rules for solving the problem are specified, one needs to be able to match up the student's behavior with these rules. This requires augmenting the rules with tests that match against the student's behavior so that the tutor can determine which rules have fired in the student's head. In the case of ambiguities, disambiguation menus must be generated from templates stored with the rules.

The product of this effort might be viewed as an "instructionless" tutor, like the nonexplanation tutors described in the evaluation section, that can be deployed in a number of tutor modalities (e.g., immediate feedback, flag, demand, no tutor). It can identify for a student where mistakes are and indicate correct courses of action. However, it can say nothing about why one action is wrong and another is correct. That awaits the construction of declarative instruction in the next stage.

Declarative Instruction

As we noted in the beginning of this article, part of the domain competence comes from declarative instruction given outside of the tutor. Successful

operation of the tutor assumes successful acquisition of this declarative knowledge. Some of this declarative instruction concerns general concepts (e.g., what an alternate interior angle is) and other instruction communicates information that will serve as the declarative basis from which production rules are compiled (e.g., one way to prove lines parallel is to show that their alternate interior angles are congruent). This declarative instruction may come in class lectures or in text material. We have often provided the student with specially written material to accompany the tutor. Recently, we have had success using hypertext facility that can be accessed in parallel with the tutor. The content of this instruction is informed by the production rules that are to be learned in the upcoming section. The instruction tries to provide examples that illustrate the rules and annotate those examples with comments that will highlight the significant aspects of the rules. A general principle in our approach to instruction is to be minimalist and not say more than is needed. This sensible approach tends not to be followed in most textbooks but is well supported by research (Reder & Anderson, 1980; Reder, Charney, & Morgan, 1986). Although this tutor-external instruction is important, of more concern to the tutor development system are the two kinds of declarative instruction delivered from within the tutor.

Error messages. When the student makes an error, one can present a message that attempts to tell the student something useful about that error. This requires writing buggy productions and attaching instruction to these productions. In general, we do not attempt to provide any deep diagnosis of the cognitive origins of the error. Rather, we simply try to explain why it is an error.

Help messages. At various points in time the student can request help or be judged in need of help, and a help message can be generated. These are generated from templates associated with the correct productions that would have fired at that point.

There are a number of issues about how to present the help messages. We have striven for a system that tries to make these messages as short and to the point as possible even if the messages sound nonhuman. Another issue concerns modality of delivery. Although we have always given these in the visual modality, we would like to add an auditory modality for instruction. Instructing in the visual modality interferes with processing of the problem, which is also in the visual modality. Instruction in the auditory modality would increase the premium placed on short messages (Laddaga, Levine, & Suppes, 1981).

Another issue concerns how to deal with students who overuse hints and, as a consequence, learn little (Shute, Woltz, & Regian, 1989). We find that linking knowledge tracing to help seeking is an effective way of dealing with hint abusers. If their progress through the tutor depends on eventually

solving the problems without help, students will not seek help unless they really need it.

Two questions remain unresolved with respect to help messages. These are whether to volunteer help to students who appear to need it and whether to present students everything in a single message or to provide a sequence of successively more explicit messages. The current hinting discipline in the tutor was designed to let students do as much as possible for themselves. This was motivated by research in psychology showing that subjects have better memory for material to the degree they participate in the generation of that material (J. R. Anderson, 1990, chap. 7). Thus, our current tutors never volunteer help and only provide help upon request.[14] Also, our current tutors use a scheme of successive hinting in which the initial help only gives a vague characterization and subsequent help messages become more specific until the student is told exactly what to do. However, these may not be the best choices. Some students stubbornly refuse to seek help even when they need it. Also, with respect to the policy of successive hinting, students are often annoyed with the vague initial messages and decide there is no point in using the help facility at all. The deployment of the tutor in courses may also influence the content of help messages. When students are using the tutor in a self-paced course or otherwise on their own, it is essential to tell students exactly what to do if necessary to allow them to proceed. In a classroom, it may be preferable for the students to interact with the teacher if they do not understand an explanation, so help messages may stop short of describing the specific action.

Deployment in the Classroom

It is interesting to consider how these tutors are actually deployed. At CMU, the programming tutors are used in self-paced learn-on-your-own environments. There are class sessions associated with the tutor, but they are largely used for administrative purposes. Students learn successfully from reading the prepared text and doing problems. They can go to the teacher if they have special problems that the tutor cannot handle, and occasionally they do. They also do larger (but still modest) projects outside of the tutor counting on competencies they have built up with the tutor.

Although this relatively teacherless and isolated model has worked reasonably well for our programming course, this model is not viable for the general deployment of tutors in public high schools. The CMU model depends on the following facts: (a) We have designed our programming tutors to deliver just the material we want to teach, (b) we have total control over our classroom, (c) we are working with relatively mature students who come

[14]This can be viewed as embedding opportunities within the tutors for students to discover the concepts for themselves.

in on their own time and are generally familiar with computers, and (d) we expect students in introductory programming courses to display their skills isolated from other students. None of these assumptions are satisfied in general. The particular nature of our programming class has made it an excellent laboratory for study of skill acquisition and issues of tutoring, but it has made it nonrepresentative for how these tutors will be deployed in other educational environments.

The typical educational environment in an American high-school mathematics class contrasts with this situation in several ways. First, there are large curriculum variations across states and school districts and smaller variations across almost every teacher within a district. As a consequence, any mathematics tutor will likely be delivering only part of the curriculum that a particular teacher delivers, so the tutor will be integrated with other classroom activities. Second, students are not mature enough to simply show up at a teacherless class and learn. They will get stuck too often in ways that the tutor cannot remediate, and discipline problems would develop. Finally, the NCTM has placed a major emphasis on teaching group problem solving in their standards (1989, 1991), so group activities will come to replace individual skill performance to varying degrees across classrooms.

Our tutors have had some successes in such classrooms. Currently, 30 classes in algebra, geometry, and Pascal programming are using a classroom of Mac II-based tutors at a local Pittsburgh high school. Students in these courses alternate between working on tutor exercises and other classroom activities, so tutor use has the flavor of going to a computer laboratory within the context of a conventional course, although students may spend as much as two thirds of their class time in the laboratory.

We are struck by the way students interact with these tutors and the consequences for class organization. Much of this was described in the early reports of Schofield and Evan-Rhodes (1989) and Wertheimer (1990), but the effects are even more striking in the current classrooms reflecting the larger class sizes and the social changes since the original classroom studies. When students are in the laboratory, they are working one-on-one with machines, but that hardly means they are working in isolation. There is a constant banter of conversation going on in the classroom in which different students compare their progress and help one another. Peer instruction is particularly key in cases where students have to adapt to a new interface feature. Information about how to use that feature propagates through the classroom much like information about how to use a new trick in a Nintendo game. We have come to realize that our tutors would be less successful if such peer assistance were not available. Peer helping may also be a good way for the helper to come to a deeper understanding of the material. An effective teacher is quite active in such a classroom, circulating about the class and providing help to students who cannot get the help they need from either the tutor or their peers. The tutor in effect becomes an assistant that can deal with the more routine learning problems, allowing the teacher to focus on the

more difficult. By means of its knowledge-tracing algorithm, it also is able to monitor separately the progress of individual students, providing a bookkeeping facility the teacher would never be able to accomplish. Teachers seem to require some time in the classroom before they appreciate the "tutor as teaching assistant" model and can use it to its maximum potential.

Students' own attitudes to the tutor classrooms are quite positive, to the point of creating minor discipline problems. Students skip other classes to do extra work on the tutor, refuse to leave the class when the period is over, and come in early. However, in net, discipline problems and class management problems are much fewer in a tutored classroom. There is a sense of satisfaction in progress and achievement. Visitors to the classroom are struck by the fact that students are absorbed in the learning tasks through the whole period. Teachers particularly remark on the success with minority students who are frequently alienated in conventional classrooms. It is our belief that students receive our tutors favorably to the degree that our tutors achieve their fundamental claim—to embody an accurate cognitive model of the details of the problem solving. If so, the interactions with the tutor are largely congruent with the student's thinking, and, when the interactions are not congruent, they point the student in the right direction. Although students do not consciously assess the system in terms of its cognitive fidelity, they are very aware of the resulting smoothness of their trajectory through learning curriculum. A sense of growing competence in a challenging problem domain is something that most people respond to positively.

We have been impressed by the relative ease of management in our tutored classrooms. If one provides teachers with a couple of weeks of familiarization with the basic software, they seem to adapt comfortably to the tutored classroom.[15] This contrasts sharply with many efforts at classroom reform, which teachers report to be quite exhausting in terms of the demands being placed on them. Our classrooms are relatively easy because the tutor is doing all of the bookkeeping and low-level instruction associated with the classroom, and the teacher can focus on giving one-on-one tutoring to students for whom the computer is not adequate. They generally find this a satisfying role and one that enhances their classroom esteem as subject-matter experts. This works because our tutors are engaging and so other students are on task when a teacher is giving one student individual attention.[16]

---

[15]However, it may take much longer before they really use the tutor to its full effectiveness. There is some evidence that achievement gains are higher the 2nd year teachers work with the tutor.

[16]This only works if our tutors are bug-free, easy to use, and allow for error recovery from things like machine crashes. Typical teachers become very frustrated when their interactions with the student must focus on the computer or the tutor, rather than the subject domain being taught.

REFLECTIONS

We have come a long way from our original goal of putting ACT* to a tough test. There certainly has been a harvest of empirical data that has played a major role in leading to the new ACT–R theory (see J. R. Anderson, 1993). We have totally abandoned our original conception of tutoring as human emulation. We now conceive of a tutor as a learning environment in which helpful information can be provided and useful problems can be selected. We are able to take actions that facilitate learning because we possess a cognitive model of where the student is in that task.

Although we have not abandoned our goals of contributing to the understanding of human cognition, we have been drawn by application to issues that are far afield. Particularly with the tutor development kit and large-scale applications, we find ourselves addressing issues of software engineering. Although we have tried to place content decisions in the hands of our "clients," we inevitably are drawn into issues about the content and purpose of high-school education. Finally, there are important social phenomena in our classrooms, critical to the success of our tutors, which we need to understand.

Many such issues may ultimately prove more important to the success of our tutors than their cognitive fidelity. However, we are impressed that 10 years later the general cognitive-modeling approach still seems viable and important in our new applications.

The Curriculum Issue

Rather than conclude this article on this self-congratulatory note, we have been persuaded to address some of the senses of unease that some people have with our efforts. There are probably many dimensions to their unease, but the reviewers of our article have gotten us to focus on issues surrounding the nature of the curriculum our tutors deliver. We have taken the liberty of quoting three of their remarks and then commenting.

> I would like to believe that a decade of research in this area has given the authors a solid perspective on what to teach, how to teach it, and how to assess the effect of that instruction. Instead of providing guidance to educators in these areas, the authors seem willing to abrogate this responsibility and to settle into the role of technologists, teaching what the current curriculum dictates regardless of the appropriateness.

We have stated strong opinions about how to assess the effect (time to reach a prescribed level of achievement), but otherwise the reviewer is correct in the

assertion.[17] Our 10 years of experience have in fact given us no basis for offering advice on some of the key issues facing the City of Pittsburgh in deciding about its mathematics curriculum. Some of these issues have been very responsibly addressed by things like the NCTM standards, and the city has adopted these. However, frankly, these standards leave open some important issues facing mathematics education in an urban setting. An example of such an issue is to what degree should the curriculum be teaching students "employable" skills (e.g., using spreadsheets) versus to what degree should the curriculum be preparing students for college mathematics (e.g., being prepared to understand the proof of the fundamental theorem of calculus). As citizens we may hold opinions on this issue, but nothing in our tutoring work informs us on the relative value of different educational goals.

> Underlying the project of tutor construction is the conviction that the subject matter can be represented as a production set. I cannot repress the suspicion that the particular choices of subject matter made by the tutor authors reflect, whether consciously or not, this conviction and, thus, that the material which lends itself less well to the theoretical framework is left unconsidered.

As the first remark, this reviewer is substantially correct in the assertion, although we again see somewhat different implications in it. Our tutors have their largest potential impact when there is a substantial production-rule component. Thus, we have stayed away from teaching simple algorithmic skills like addition and focused on high-school mathematics because it seemed that this is where the largest impact would be. Some "advanced" domains are largely declarative. So we have considered a tutor for cognitive psychology but have concluded that the body of knowledge typically taught in an undergraduate course is largely factual, and very limited inferential chains are required. (We think, by the way, in our role as citizens of the field and as authors of a cognitive psychology text, that this reflects a serious indictment of instructional goals in the field.) It is also the case that developing cognitive modeling is an expensive enterprise (we estimate 10 hr or more per production rule), and it may not be economical or feasible to model all the competence involved within more advanced areas of mathematics.

However, although there are these practical limitations, we do not believe that there are any fundamental limitations of the approach. For instance, we are currently working on developing a tutor for exploration and discovery in the context of geometry because these are typically thought to be skills outside the domain of our tutor. Although we do not have any educational results, we

[17]However, assenting to the reviewer's comment requires interpreting "current curriculum" to Pittsburgh's algebra and geometry curriculum that follows, and in some way goes beyond, the NCTM standards.

can report that the skills are perfectly capable of being modeled within a production-system framework (as indeed earlier research would have indicated; e.g., Klahr & Carver, 1988).

> I find the production-rule-based approach to subject matter ... well ... less than fun. It would be hard for me to believe that a student would choose to study geometry on his or her own—would go home and start playing with geometric shapes or cutouts or models or whatever—based on experience with these tutors.

We think it is easy to underestimate the motivational gains produced by the simple experience of learning achievement. The principal reason for the enthusiasm for our tutors within the Pittsburgh Public School System is motivational gains, not achievement gains. Perhaps our favorite anecdote is about one student in a school in another state that had the LISP tutor. The student, frustrated by restrictive access to the LISP tutor, deliberately induced a 2-day suspension by swearing at a teacher. He used those 2 days to dial into the school computer from his home and complete the lesson material on the LISP tutor.

Although the issue of the content of the curriculum is essential, learning achievement is a very empowering experience. Thinking back on one's own learning experiences and the environment one learned in, it is easy to take learning for granted and only focus on what is being learned. The fact that something will be learned cannot be taken for granted in many American schools.

ACKNOWLEDGMENTS

We thank Lael Schooler for his comments on this article.

Over its 10-year history, the research reported in this article has been supported by Contracts MDA 903–85–K–0343 and MDA 903–89–K–0190 from the Army Research Institute; a grant from the Carnegie Corporation; Grant Nos. MDR–84–70337, IST–83–18629, MDR–87–15890, MDR–89–54745, and MDR–92–53161 from the National Science Foundation; and Contracts N00014–84–K–0064, N00014–87–0103, and N00014–91–J–1597 from the Office of Naval Research.

REFERENCES

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

Anderson, J. R. (1990). *Cognitive psychology and its implications.* New York: Freeman.

Anderson, J. R. (Ed.). (1993). *Rules of the mind.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J. R., Bellezza, F. S., & Boyle, C. F. (1993). The geometry tutor and skill acquisition. In J. R. Anderson (Ed.), *Rules of the mind* (pp. 165–181). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Anderson, J. R., Boyle, C. F., Farrell, R., & Reiser, B. J. (1987). Cognitive principles in the design of computer tutors. In P. Morris (Ed.), *Modeling cognition* (pp. 93–134). New York: Wiley.

Anderson, J. R., Boyle, C. F., & Yost, G. (1986). The geometry tutor. *The Journal of Mathematical Behavior, 5,* 5–19.

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

Anderson, J. R., Conrad, F., Corbett, A. T., Fincham, J. M., Hoffman, D., & Wu, Q. (1993). Computer programming and transfer. In J. R. Anderson (Ed.), *Rules of the mind* (pp. 205–233). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Anderson, J. R., Farrell, R., & Sauers, R. (1984). Learning to program in LISP. *Cognitive Science, 8,* 87–130.

Anderson, J. R., Greeno, J. G., Kline, P. J., & Neves, D. M. (1981). Acquisition of problem-solving skill. In J. R. Anderson (Ed.), *Cognitive skills and their acquisition* (pp. 191–230). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Anderson, J. R., & Pelletier, R. (1991). A development system for model-tracing tutors. In L. Birnbaum (Ed.), *Proceedings of the International Conference of the Learning Sciences* (pp. 1–8). Charlottesville, VA: AACE.

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

Anderson, L. W., & Burns, R. B. (1987). Values, evidence, and mastery learning. *Review of Educational Research, 57,* 215–223.

Block, J. H. (1971). *Mastery learning.* New York: Holt, Rinehart & Winston.

Bloom, B. S. (1984). The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher, 13,* 4–16.

Brown, J. S. (1985). Idea amplifiers: New kinds of electronic learning. *Educational Horizons, 63,* 108–112.

Brown, J. S., Collins, A., & Duguid, P. (1989). Situated cognition and the culture of learning. *Educational Researcher, 18*(1), 32–41.

Bunderson, C. V., & Faust, G. W. (1976). Programmed and computer-assisted instruction. In N. L. Gage (Ed.), *The psychology of teaching methods: 75th Yearbook of the National Society for the Study of Education.* Chicago: University of Chicago Press.

Cognition and Technology Group at Vanderbilt. (1990). Anchored instruction and its relationship to situated cognition. *Educational Researcher, 19*(6), 2–10.

Collins, A., & Brown, J. S. (1987). The computer as a tool for learning through reflection. In H. Mandl & A. M. Lesgold (Eds.), *Learning issues for intelligent tutoring systems* (pp. 1–18). New York: Springer-Verlag.

Collins, A., Brown, J. S., & Newman, S. E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L. B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser* (pp. 453–494). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

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

Corbett, A. T., & Anderson, J. R. (1992). Student modeling and mastery learning in a computer-based programming tutor. In C. Fiasson, G. Gauthier, & G. McGalla (Eds.), *Proceedings of the Second International Conference on Intelligent Tutoring Systems* (pp. 413–420). Berlin: Springer-Verlag.

Corbett, A. T., Anderson, J. R., & O'Brien, A. T. (1995). Student modeling in the ACT programming tutor. In P. Nichols, S. Chipman, & R. Brennan (Eds.), *Cognitively diagnostic assessment* (pp. 19–41). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Gagné, R., & Briggs, L. J. (1974). *Principles of instructional design.* New York: Holt, Rinehart & Winston.

Goldenson, D. R. (1989a). The impact of structure editing on introductory computer science education: The results so far. *SIGCSE Bulletin, 21,* 26–29.

Goldenson, D. R. (1989b). Teaching introductory programming methods using structure editing: Some empirical results. In W. C. Ryan (Ed.), *Proceedings of the National Educational Computing Conference 1989* (pp. 194–203). Eugene: University of Oregon, International Council on Computers in Education.

Guskey, T. R. (1987). Rethinking mastery learning reconsidered. *Review of Educational Research, 57,* 225–229.

Guskey, T. R., & Gates, S. (1986). Synthesis of research on the effects of mastery learning in elementary and secondary classrooms. *Educational Leadership, 43,* 73–80.

Key Curriculum Press. (1991). *Geometer's sketchpad* [Computer software]. Berkeley, CA: Author.

Klahr, D., & Carver, S. M. (1988). Cognitive objectives in a LOGO debugging curriculum: Instruction, learning and transfer. *Cognitive Psychology, 20,* 362–404.

Koedinger, K., & Anderson, J. R. (1993). Effective use of intelligent software in high school math classrooms. In *Artificial intelligence in education: Proceedings of the World Conference on AI in Education* (pp. 241–248). Charlottsville, VA: AACE.

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

Kulik, C., Kulik, J., & Bangert-Downs, R. (1986). *Effects of testing for mastery on student learning.* Paper presented at the annual meeting of the American Educational Research Association, San Francisco, CA.

Laddaga, R., Levine, A., & Suppes, P. (1981). Studies of student preference for computer-assisted instruction with audio. In P. Suppes (Ed.), *University-live computer-assisted instruction at Stanford: 1968–1980* (pp. 399–430). Stanford, CA: Institute for Mathematical Studies in the Social Sciences.

Lave, J., & Wenger, E. (1990). *Situated learning: Legitimate peripheral participation.* Palo Alto, CA: Institution for Research in Learning.

Lewis, M. W. (1989). *Developing and evaluating the CMU algebra tutor: Tension between theoretically driven and pragmatically driven design.* Paper presented at the annual meeting of the American Educational Research Association, San Francisco, CA.

Matz, M. (1982). Towards a process model for high school algebra errors. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems* (pp. 25–50). New York: Academic.

McKendree, J. (1990). Effective feedback content for tutoring complex skills. *Human–Computer Interaction, 5,* 381–413.

McKendree, J. E. (1986). *Impact of feedback content during complex skill acquisition.* Unpublished doctoral dissertation, Carnegie Mellon University, Pittsburgh, PA.

Milson, R., Lewis, M. W., & Anderson, J. R. (1990). The teacher's apprentice project: Building an algebra tutor. In R. Freedle (Ed.), *Artificial intelligence and the future of testing* (pp. 53–71). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

National Council of Teachers of Mathematics Commission on Standards for School Mathematics. (1989). *Curriculum and evaluation standards for school mathematics.* Reston, VA: Author.

National Council of Teachers of Mathematics. (1991). *Professional standards for teaching mathematics.* Reston, VA: Author.

Polson, P., & Kieras, D. E. (1985). A quantitative model of learning and performance of text editing knowledge. In L. Bormann & B. Curtis (Eds.), *Proceedings of CHI '85 Human Factors in Computing Systems Conference* (pp. 207–212). New York: Association for Computing Machinery.

Porter, D. (1961). *An application of reinforcement principles to classroom teaching.* Cambridge, MA: Harvard University, Graduate School of Education, Laboratory for Research in Instruction.

Reder, L. M., & Anderson, J. R. (1980). A comparison of texts and their summaries: Memorial consequences. *Journal of Verbal Learning and Verbal Behavior, 198,* 121–134.

Reder, L. M., Charney, D. H., & Morgan, K. I. (1986). The role of elaborations in learning a skill from an instructional text. *Memory and Cognition, 14,* 64–78.

Scheines, R., & Sieg, W. (1993). *An experimental comparison of alternative proof construction environments* (Tech. Rep. No. CMU–PHIL–40). Pittsburgh, PA: Carnegie Mellon University, Department of Philosophy.

Schofield, J. W., & Evan-Rhodes, D. (1989). Artificial intelligence in the classroom: The impact of a computer-based tutor in teachers and students. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Artificial intelligence and education* (pp. 238–243). Amsterdam: IOS.

Shephard, L. A. (1992). Psychometricians' beliefs about learning. *Educational Researcher, 21,* 2–16.

Shute, V. J., Woltz, D. J., & Regian, J. W. (1989). An investigation of learner differences in an ITS environment: There is no such theory as a free lunch. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Artificial intelligence and education* (pp. 260–266). Amsterdam: IOS.

Singley, M. K. (1986). *Developing models of skill acquisition in the context of intelligent tutoring systems.* Unpublished doctoral dissertation, Carnegie Mellon University, Pittsburgh, PA.

Singley, M. K., & Anderson, J. R. (1989). *The transfer of cognitive skill.* Cambridge, MA: Harvard University Press.

Singley, M. K., Anderson, J. R., Gevins, J. S., & Hoffman, D. (1989). The algebra word problem tutor. In D. Bierman, J. Breuker, & J. Sandberg (Eds.), *Artificial intelligence and education* (pp. 267–275). Amsterdam: IOS.

Slavin, R. E. (1987). Taking the mystery out of mastery: A response to Guskey, Anderson, and Burns. *Review of Educational Research, 57,* 231–235.

Sleeman, D. H., & Brown, J. S. (1982). *Intelligent tutoring systems.* London: Academic.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive Science, 12,* 257–286.

Wertheimer, R. (1990). The Geometry Proof tutor: An "intelligent" computer-based tutor in the classroom. *Mathematics Teacher,* 308–313.

Whitehead, A. N. (1929). *The aims of education.* New York: Macmillan.