Intelligent Tutoring Systems

Handbook of Human-Computer Interaction, Second, Completely Revised Edition in M. Helander, T. K. Landauer, P. Prabhu (Eds), Elsevier Science B. V., ©1997, Chapter 37

Albert T. Corbett and Kenneth R. Koedinger
Human Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

John R. Anderson
Department of Psychology and Computer Science
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA

Computers have been employed to achieve a variety of educational goals since the early 1960s. Some of these goals include automated testing and routine drill and practice tasks that had been mechanized with earlier technologies, as far back as the thirties. Other computer assisted instructional programs engage the students in challenging and entertaining reasoning tasks and capitalize on multimedia capabilities to present in formation (cf. Larkin and Chabay, 1992). Computer- based instruction has successfully penetrated all education and training markets: home, schools, universities, business, and government, but remains far from the modal educational experience.

In the early 1970s a few researchers defined a new and ambitious goal for computer-based instruction. They adopted the human tutor as their educational model and sought to apply artificial intelligence techniques to realize this model in “intelligent” computer- based instruction. Personal human tutors provide a highly efficient learning environment (Cohen, Kulik and Kulik, 1982) and have been estimated to increase mean achievement outcomes by as much as two

standard deviations (Bloom, 1984)[1]. The goal of intelligent tutoring systems (ITSs) would be to engage the students in sustained reasoning activity and to interact with the student based on a deep understanding of the students behavior. If such systems realize even half the impact of human tutors, the payoff for society promised to be substantial.

Seminal intelligent tutoring efforts are well documented in two classic books, Sleeman and Brown (1982) and Wenger (1987), and research efforts over the ensuing twenty-five years have yielded some notable successes in achieving the promise of intelligent tutoring (Lesgold, Eggan, Katz and Rao, 1992; Koedinger, Anderson, Hadley and Mark, 1995). In this chapter we abstract a set of design recommendations from this extended research effort. The chapter contains three main sections. The first provides an over view of intelligent tutoring systems. The second section provides a prescription for ITS design and development methods. The third section addresses several issues concerning ITS design principles.

37.2 Overview of Intelligent Tutoring Systems

This section provides an overview of intelligent tutoring systems. We begin with a comment on competing research goals in the field, followed by descriptions of two successful systems: (I) our own PUMP Algebra Tutor Project and (2) the SHERLOCK project (Lesgold, Eggan, Katz and Rao, 1992). These two systems are being deployed in real-world educational environments with substantial success. We briefly describe the underlying theory and implementation of each, to motivate later discussions. We follow with a description of the standard components in an intelligent tutoring system and conclude with a discussion of HCI assessment issues unique to educational software. Readers who are well-versed in the field may wish to skip over this overview to the following sections on design methods and principles.

37.2.1 Research Goals: Al vs. Education

While there has been a sustained research effort in the application of artificial intelligence to education over the past twenty-five years with some notable success stories, intelligent tutoring has had relatively little impact on education and training in the world. There are several reasons for this lack of penetration. Intelligent tutoring systems are expensive to develop and until relatively recently, the necessary computing power was expensive to deploy. However, we believe that an even more important reason can be traced to the ontogeny of the field and the consequences for soft ware evaluation. The creative vision of intelligent computer tutors has largely arisen among artificial intelligence researchers rather than education specialists. Researchers recognized that intelligent tutoring systems are a rich and important natural environment in which to deploy and improve Al algorithms. We have outlined elsewhere a variety of consequences that de rive from this history in Al, (Anderson and Corbett, 1993) but the bottom line is that intelligent tutoring systems are generally evaluated according to artificial intelligence criteria - the coverage of the systems in interpreting and responding to student behaviors -rather than with respect to a cost/benefit analysis educational effectiveness.

[1] Evaluations of instructional effectiveness employ a variety of learning time and achievement measures. The conventional method for comparing educational manipulations is to convert effective sizes into standard deviation units, that is, to subtract the mean of a control group from the mean of the the experimental group and divide by the standard deviation of the control group. An effect of 2 standard deviations indicates that 98% of the experimental group performed as well as the top 50% of the control group. This is not a perfect measure, since apparent effect sizes depend on sample variability.

The first intelligent tutoring program, SCHOLAR (Carbonell, 1970) merits special recognition and serves to exemplify this pattern. This program attempted to engage the student in a mixed-initiative dialogue on South American geography. The program and student communicated through a sequence of natural language questions and answers. The tutor could both ask and answer questions and keep track of the ongoing dialogue structure. This tutor was constructed around a semantic network model of domain knowledge. Such network models of conceptual knowledge were revolutionizing our understanding of question answering and inferential reasoning in cognitive science (cf. Anderson and Bower, 1973; Collins and Loftus, 1975) and re main the modal model of conceptual knowledge today (cf. Anderson, 1983). However, the effort to sustain a dialogue revealed the importance of unexplored issues in dialogue structure and pragmatic reasoning. This fed into an interesting research program and successive dialogue tutors (Stevens and Collins, 1977; Collins and Steven, 1991; Woolf and McDonald, 1984), but in the end the fascinating and challenging issues in natural language comprehension took precedence over research in how to deploy dialogue tutors effectively.

While there has been a rich intellectual history in intelligent tutoring, we believe that for intelligent tutors to seriously penetrate the educational/training system, the evaluative focus must begin to shift to educational impact and away from artificial intelligence sufficiency. This has begun to happen, but at each of the two most recent International Conferences on Intelligent Tutoring Systems (Frasson, Gauthier and McCalla, 1992; Frasson, Gauthier and Lesgold, 1996), only 25% of non-invited papers included empirical evaluations of any sort. Among these empirical studies only about one in ten (i.e., 2.5% of all papers) assessed the effectiveness of a computer-based learning environment by comparing SW- dent performance to other learning environments. We believe the emphasis on educational impact must permeate all stages of ITS development, deployment and assessment. After twenty five years we can frame many important questions in effective intelligent tutoring, but can provide only preliminary answers.

37.2.2 The PUMP Algebra Tutor (PAT) Project

We have been developing and using intelligent tutoring systems for mathematics and programming for a little over a decade (Anderson, Corbett, Koedinger and Pelletier, 1995). Four years ago we partnered with the Pittsburgh School District to develop an intelligent tutoring system for introductory algebra. Events in mathematics education at the national level coalesced with the-situation in the Pittsburgh schools to make this an opportune target. The recent National Council of Teachers of Mathematics (1989) curriculum standards advocate mathematics for all students. These standards recommend an algebra curriculum that is relevant for students bound for college or directly for the workplace. They de-emphasize symbol manipulation and emphasize reasoning about authentic problem situations with multiple representations, including symbolic expressions, natural language, graphs and tables. In emphasizing relevance, the standards also advocate reasoning about mathematics in the context of modern computational tools, e.g., graphics calculators and spreadsheets. These recommendations directly speak to mathematics education in Pittsburgh, which is typical of a city its size. Only about two-thirds of high school freshmen take algebra I and among students who enter the academic mathematics track (algebra I, geometry, algebra II, precalculus), about 40% drop out each year. Mathematics achievement levels are appreciably below the national average.

A number of mathematics teachers and curriculum supervisors in the school district are active in mathematics curriculum reform at the national level and contributed to the National Council of

Teachers of Mathematics standards that emphasized academic mathematics for all students. These local mathematics educators formed the Pittsburgh Urban Math Project (PUMP) to develop a new algebra curriculum consistent with the new standards. This curriculum became the starting point for the PUMP Algebra Tutor (PAT). PAT is an algebraic problem solving environment. Each task presents a problem solving situation that describes the relationships among two or three quantities and presents from three to six specific questions to answer. The students are asked to represent the quantities as spreadsheet columns, answer the questions in the spreadsheet rows, induce an algebraic description of the relationships and graph these relationships.

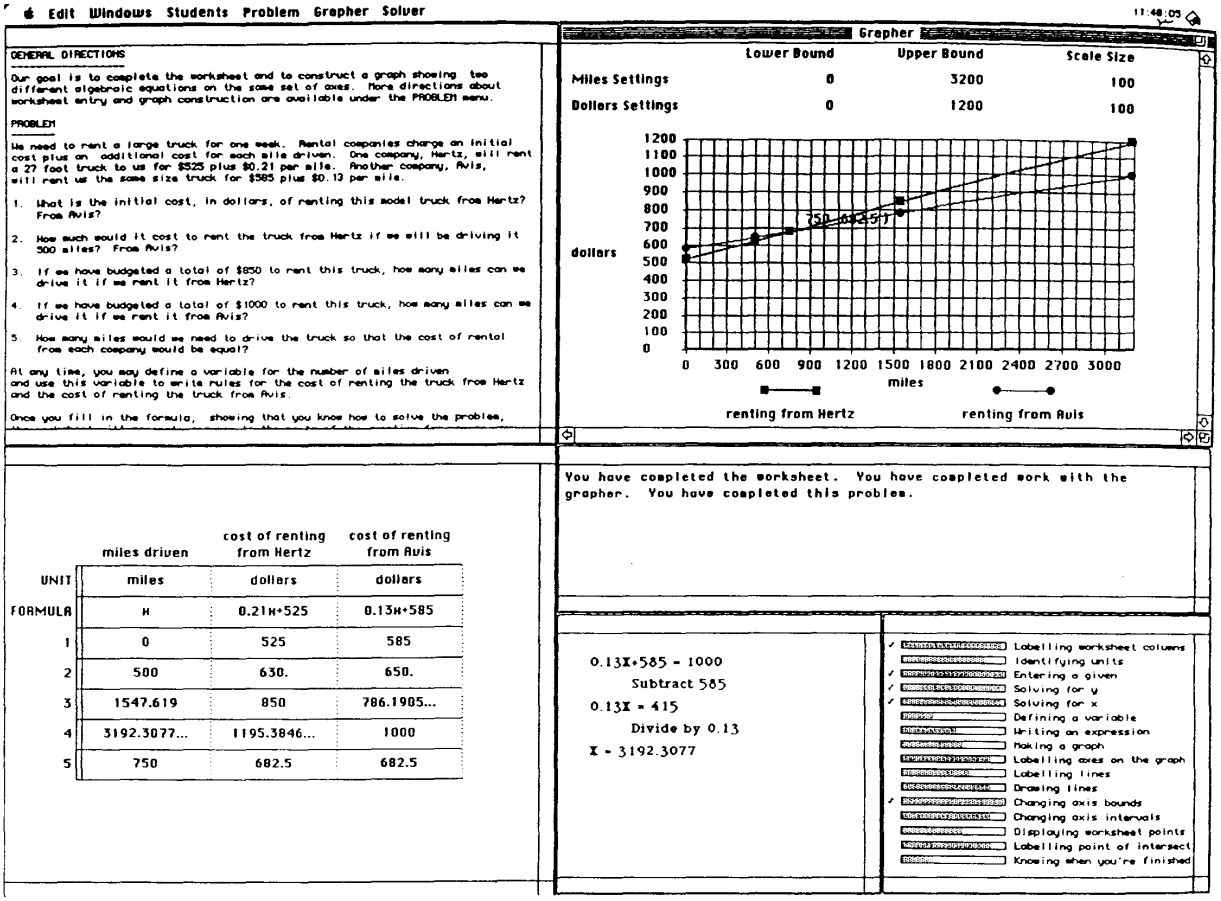

Figure 1. The PUMP algebra tutor.

Figure 1 displays the algebra word problem tutor screen in the middle of an exercise. The exercise description appears in a window at the upper left. In this exercise the student is solving an exercise that calls for a system of two equations. In the lower left, the student has constructed the table that exemplifies the relation ships among the quantities in the exercise. This table is characteristic of a typical computer spreadsheet application. The columns represent the quantities and the rows represent different example values. In this exercise, the first column represents distance traveled arid the second and third columns represent the cost of renting a car from two different agencies. The student has labeled the quantities at the top of

each column and indicated the units in the first row. The student also enters a mathematical variable, e.g., x, into the second row of the first column to formalize this quantity. The student has constructed mathematical expressions for deriving each of the other two quantities with this variable (0.2 lx + 525 and 0.l3x ± 585) and entered them into the second and third columns. Finally, the student enters example values into the remaining rows in response to specific queries in the exercise description. Having constructed this table, the student also graphs the equations in the upper right. The student generates an appropriate scale and label for each axis, and identifies the intersection of the two lines. Thus, the student gains experience with alternative representations: the table, the symbolized expression and the graphical representation.

PAT's first classroom piloting came in 1992-1993 in Pittsburgh's Langley High School. Students worked with the tutor in about 25 algebra I class periods (out of almost 180) that first year. In 1993-1994 all incoming freshman in Langley high school entered the academic mathematics track via the PUMP algebra course and the PAT project expanded to two other Pittsburgh high schools. In this second year, students spent about 40 class periods with the tutor. By 1995-1996, approximately 750 students in six area high schools worked through the PUMP algebra course and tutor. In 1993- 1994 and 1994-1995 year-end assessments, PUMP algebra students performed 100% better than comparable traditional algebra I students on assessments of reasoning among multiple representations in authentic problem solving situations (a one-standard deviation improvement). PUMP algebra students also performed 10-15% better on standardized mathematics tasks, e.g., Math SAT exercises, that are de-emphasized in the PUMP/PAT curriculum.

Note that we cannot separate the impact of the intelligent tutoring system from the impact of the re formed curriculum. In the past we have achieved similar one-standard deviation effects of intelligent tutoring with a constant curriculum (Anderson, Boyle, Corbett and Lewis, 1990; Corbett and Anderson, 1991; Koedinger and Anderson, 1993a). However, the more important point is that this project that began with an assessment of educational needs rather than concern with artificial intelligence innovation and resulted in an integrated package that yielded a one-standard deviation benefit compared to traditional practice.

37.2.3 Cognitive Theory

The tutors reflect the ACT-R theory of skill knowledge (Anderson, 1993). This theory assumes a fundamental distinction between declarative knowledge and procedural knowledge. Declarative knowledge is factual or experiential. For example, the following sentence and example on equation solving in an algebra text would be encoded declaratively:

When the quantities on both sides of an equation are divided by the same value, the resulting quantities are equal.

For example, if we assume 2X = 12, then we can divided both sides of the equation by 2, and the two resulting expressions will be equal, X = 6.

ACT-R assumes that skill knowledge is encoded initially in declarative form through experiences such as reading. While such declarative knowledge is a pre requisite of domain expertise, efficient problem solving requires the encoding of goal-oriented procedural knowledge. ACT-R assumes that early in practice the student solves problems by applying general procedural rules to the domain-specific knowledge. As a consequence of this early activity, domain procedural

knowledge is acquired. With subsequent practice, both declarative and procedural knowledge are strengthened so that performance grows more rapid and reliable. Like many cognitive theories (cf. Kieras and Bovair, 1986; Newell, 1990; Reed, Dempster and Ettinger, 1985) ACT-R assumes that procedural knowledge can be represented as a set of independent production rules that associate problem states and problem-solving goals with actions and consequent state changes. The following goal-oriented production can be derived from the declarative example above through practice in solving equations:

IF the goal is to solve an equation for variable X and the equation is of the form aX = b, THEN divide both sides of the equation by a to isolate X.

PAT contains many rules, for labeling table columns, entering problem givens and computing results, inducing algebraic expressions from examples, solving equations, labeling and scaling graphs and placing points. We call the set of programming rules built into the tutor the ideal student model, because it embodies the knowledge that the student is trying to acquire. The ideal model plays two roles in our tutors, as described below. First, it allows the tutor to solve exercises step- by-step along with the student in a process we call model tracing. Second, it is used to trace each student's growing knowledge in a process we call know! edge tracing.

Model-Tracing Tutors. A model-tracing tutor solves each problem step-by-step along with the student, pro viding assistance as needed. The goal is to follow each student's individual solution path (sequence of problem solving actions) through a problem space that may contain thousands of solution paths. In each problem solving cycle the student selects an interface element to work on, (e.g., a spreadsheet cell), and performs a problem solving action, (e.g., typing a numeric value). Each of these interface elements is linked to an internal representation of a problem solving goal with links to relevant information in the current problem solving state. The tutor applies its production rules to the goal the student implicitly selects and generates a set of one or more applicable rules that satisfy the goal. The student's action is compared to the actions this set of applicable rules would generate. If the student action matches a production rule action it is assumed that the student has fired the same cognitive rule and the actions are carried out. If not, the tutor reports that it does not recognize the student's action.

Tracing the student's step-by-step solution enables the tutor to provide individualized instruction in the problem solving context. Prototypically our tutors provide immediate feedback on each problem solving action: recognizably correct actions are accepted and unrecognized actions are rejected. Our tutors do not try to diagnose student misconceptions and do not automatically give problem solving advice. Instead, they allow the student maximum opportunity to reason about the current problem state. The tutors do provide a feedback message if the student appears confused about the nature of the current problem state or a problem solving action. For example, the tutor will point out if the student has entered the right answer for one cell in the wrong cell of a table. The tutors provide goal-directed problem solving advice upon request. We recognize three general levels of advice: a reminder of the current goal, a general description of how to achieve the goal, and a description of exactly what problem solving action to take. Each of these three levels may be represented by multiple messages.

The ideal student model is also used to trace the student's growing knowledge state and to implement a variation of mastery learning. Each curriculum section introduces a set of rules in the ideal model. The tutor estimates the probability that the student knows each production based on the student's action at each opportunity to apply the rule in a process we call knowledge tracing. This process employs a simple Bayesian decision process described

elsewhere (Corbett and Anderson, 1995b). In mastery learning, students continue solving problems in a section until the probability that the student knows each rule has reached near certainty.

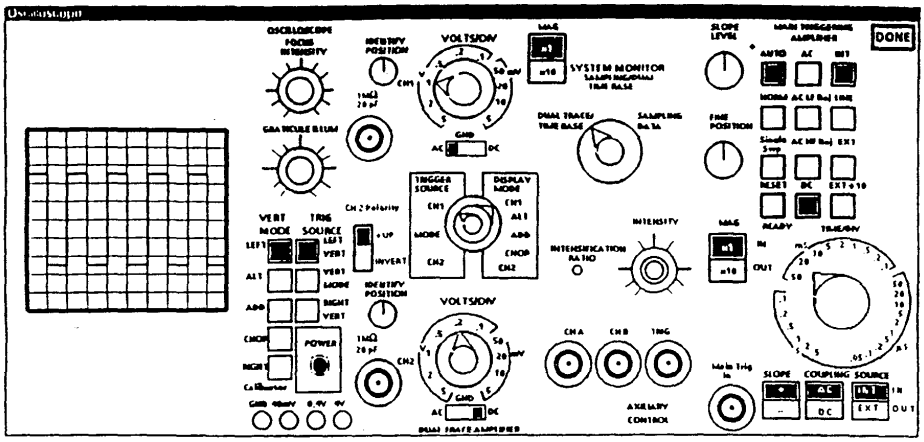

Figure 2A. An external control panel of the F15 Avionics test station. From A.Lesgold, S. Lajoie, M. Bunzo and G. Eggan, 1992. Reprinted by permission.

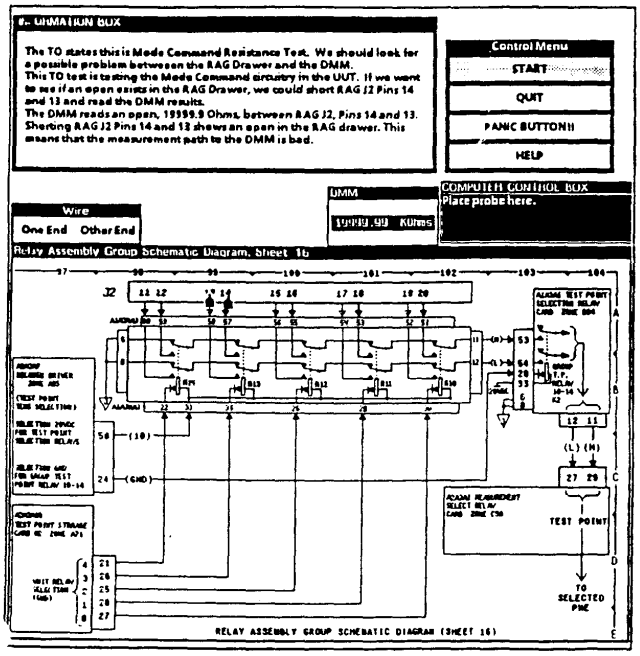

Figure 2B. The SHERLOCK interface displaying a schematic representation of an internal component of the F15 avionics test station. *From A.Lesgold, S. Lajoie, M. Bunzo and G. Eggan, 1992. Copyright 1992 by Lawrence Erlbaum Associates. Reprinted by permission.*

37.2.4 The SHERLOCK Project

SHERLOCK is a practice environment for electronics troubleshooting commissioned by the Air Force (Lesgold, Laioie, Bunzo and Eggan, 1992; Katz and Lesgold, 1993). Where PAT represents one of the mainstreams in intelligent tutoring, mathematics problem solving (Sleeman, Kelly, Martinak, Ward and Moore, 1989;) SHERLOCK represents a second main stream, diagnosis (Brown, Burton and Zdybel, 1973; Clancey 1982; Gitomer, Steinberg and Mislevy, 1995; Tenney and Kurland, 1988). SHERLOCK is designed to give technicians practice in troubleshooting the F-IS Avionics Test Station. The test station is itself an electronic system for diagnosing failures in F-IS electronic equipment. However, the test station may malfunction and this poses an interesting problem. While there are detailed procedures for using the test station to diagnose el failures in the F-IS, there are no fixed procedures for troubleshooting the test station. Since the test station breaks down infrequently, there is limited opportunity for technicians to learn troubleshooting heuristics on the job. Many of the technicians are in the relevant maintenance position for a limited time, so there is scant opportunity to build up expertise. SHERLOCK addresses this problem by providing ample troubleshooting opportunities.

In a characteristic simple problem solving scenario the trainee is informed that the test station was used to measure an F15 electronic component and the meter reading indicated component malfunction. However, when the suspect F15 component was replaced the meter reading still indicated component malfunction, suggesting that the test station itself is malfunctioning. The problem solving goal is to track down the source of the test station malfunction. SHERLOCK presents a visually realistic simulation of the external control panel of the avionics test station as shown in Figure 2A and the student can manipulate the controls with the mouse. The internals of the test station are represented by schematic diagrams. Figure 2B displays the schematic for one test station component. The student can take measurements by indicating the points on the schematic to attach' leads and the student can manipulate components, e.g., tightening connections to or 'swapping-in' a replacement for suspect components in the schematic diagram. The student communicates an intended action category and test station component through a set of hierarchical menus.

Unlike PAT, SHERLOCK generally does not pro vide immediate feedback on problem solving actions. Like PAT, SHERLOCK provides advice on problem solving steps upon student request. Four types of feed back are available: (1) advice on the what test action to carry out and how, (2) advice on how to read the out come of the test, (3) advice on what conclusion can be drawn from the test and (4) advice on what option to pursue next.

The cognitive theory underlying SHERLOCK is compatible with ACT-R, but the tutor's internal representation of domain knowledge is quite different. Any problem solving task can be characterized as a problem solving "space" containing problem solving states that are linked by problem solving actions. PAT dynamically generates this problem space as rules in the cognitive model are applied in problem solving. In SHERLOCK, this problem solving space is stored as an explicit data structure. Only abstract problem states and problem solving actions are explicitly represented. Details concerning the device and concrete actions are embedded in executable code. Students navigate through this problem solving space with the set of hierarchical menus mentioned above. Student actions are interpreted with respect to information stored in each state.

Independent field tests of SHERLOCK were conducted by the Air Force, comparing three groups: (1) an experimental group of relative novices who completed 20-25 hours of coached practice with SHERLOCK, (2) a control group of similar novices who continued with usual work activities, and (3) a group of experienced technicians. The two novice groups completed pretests and posttests of their diagnostic skills while the expert group completed a single test. The two novice groups performed equivalently on the pretest, while the expert technicians scored about 40% higher. On the posttest the experimental group scores rose about 35%, essentially overtaking the expert technicians, while the control scores did not rise significantly. A regression analysis indicates that the 20-25 hours of practice with SHERLOCK had an equivalent impact to 4 years of on- the-job experience. SHERLOCK achieves this stunning result in two ways, by affording the opportunity for extensive practice and by creating educationally effective instructional conditions.

Despite these impressive results, this field test was essentially a formative evaluation, designed to examine and redesign the system where necessary (Lesgold, Eggan, Katz and Rao, 1992). This analysis led to a series of changes embodied in the SHERLOCK II system. The schematic representations of the test station internal components were replaced by faithful visual representations. In SHERLOCK I measurement readings were stored in tables rather than computed dynamically and students could only take measurements that were judged to be reasonably related to the problem situation. In SHERLOCK II measurement readings are generated dynamically and students can perform unexpected measurements. An analysis of problem solving procedures across SHERLOCK's problem situations led to a hierarchical model of progressively more abstract methods. This representation is used in SHERLOCK II to provide advice at more abstract levels to mediate feedback. Finally, a "reflective follow- up" was added to each problem solving task in which students can replay their problem solving actions, explore the problem state at each step and obtain advice on optimal strategy.

37.2.5 Intelligent Tutoring Architecture

While the seminal intelligent tutoring project SCHOLAR employed a conversational dialogue the overwhelming majority of intelligent tutoring systems employ a different tutoring model as exemplified in

PAT and SHERLOCK: coached practice. In these environments instruction is delivered in the context of students engaged in problem solving tasks. There are pedagogical reasons for the dominance of this paradigm: It embodies the "learning-by-doing" model of cognitive skill acquisition. There are also computational advantages associated with this paradigm. Many problem solving domains provide a problem solving formalism that is easier to parse than natural language. As might be expected, the earliest problem solving tutors focused on such formal domains as mathematics (Brown, 1983; Burton and Brown, 1982), programming (Johnson and Soloway, 1984; Miller, 1979; Westcourt, Beard and Gould, 1977; cf. Brusilovsky, 1995) and electronics (Brown, Burton and Zdybel, 1973). As recently as the 1992 International Conference on Intelligent Tutoring Systems (Frasson, Gauthier and McCalla, 1992) 50% of the presentations were drawn from these three areas, although the proportion of presentations in these areas dropped substantially by the 1996 conference (Frasson, Gauthier and Lesgold, 1996). In addition, a problem solving task constrains the reasoning space for both the student and tutor. In a dialogue if the tutor does not understand a question or know the answer, the dialogue flounders. Pragmatically, in problem solving, the tutor can always give advice even if it cannot interpret the student's action. Moreover, there is less of a premium on a "natural" interaction style in coached practice than in a dialogue. Our discussion of intelligent tutoring

systems will focus on coached practice systems. The classic intelligent tutoring architecture consists of four components, as displayed in Figure 3: (1) a task environment, (2) a domain knowledge module, (3) a student model and (4) a pedagogical module. An ex tended discussion of these components can be found in Poison and Richardson (1988). Students engage in problem solving activities in the problem solving environment. These actions are evaluated with respect to the domain knowledge component, typically expert problem solving knowledge. Based on this evaluation a model of each student's knowledge state is maintained. Finally, the pedagogical module delivers instructional actions based on the evaluation of student actions and on the student model.

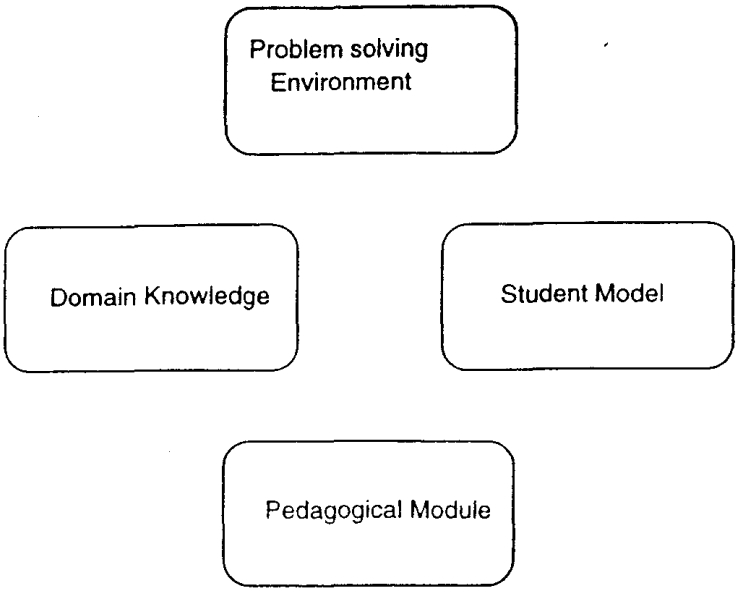

Figure 3. Components of an intelligent tutor architecture.

The Problem-Solving Environment. This component of the tutor defines the problem solving activities in which the student is engaged. At minimum it consists of an editor that accepts and represents student actions. For example, in the case of programming tutors, the interface may be a text editor (Johnson and Soloway, 1984), structure editor (Anderson and Reiser, 1985) or graphical editor (Reiser, Kimberg, Lovett and Ranney, 1992). As can be seen in Figure 1, the PAT interface contains a spreadsheet and graphing window. The task environment can also contain powerful device or sys tem simulations. For example, the early system STEAMER (Williams, Hollan and Stevens, 1981) simulated a shipboard steam propulsion plan. SMITHTOWN (Shute and Glaser, 1990) simulates economic market forces and, of course, SHERLOCK simulates an avionics test station. There are a variety of well-recognized reasons for structuring training around device and system simulations. The actual systems may be unavailable for training purposes and novice errors with actual systems may cause expensive damage or dangerous situations. A simulation's response to student actions provides a form of

feedback, independent of tutorial advice, that enables the student to reason about the problem solving situation. Finally, simulations can yield substantial savings in learning time. In the case of SHERLOCK for example, students do not have to spend time physically connecting lead wires to take measurements.

There is a strong consensus on two principles of problem solving interface design.

(1) The ITS problem solving environment should approximate the real world problem solving environment.

(2) The ITS problem solving environment should facilitate the learning process.

The first principle is implicit in the NCTM recommendations that students use real world computational tools in learning mathematics. When we began developing intelligent tutors over a decade ago we thought of the problem solving interface primarily as a vehicle for learning an abstract skill. We have come to recognize that at least at first, skill acquisition is closely tied to overt problem solving actions, and that even the expression of abstract knowledge requires the learning of many concrete interface activities. Consequently, similarity of learning environments and real world environments affords greater transfer of learning. This recognition has led to a strand of research in which we are building tutors for commercially avail able mathematical software applications (Ritter and Koedinger, 1995). This research is in an early stage but it may lead to a shift in the deployment of intelligent tutoring technology. Instead of acquiring a tutor to learn a skill, the user will acquire a software application with “lightweight tutoring agents” that can be engaged to support learning in the environment.

On the surface the second principle is in conflict with the first; it recommends deviating from the work place problem solving environment to increase learning rate. There are several ways in which the interface can be modified to support learning. First, it is possible to abstract away irrelevant aspects of the problem solving task. In the case of device simulations this means eliminating potentially time consuming physical actions such as attaching electronic leads (Lesgold, Eggan, Katz and Rao, 1992). A second modification is to provide augmented feedback on device states or problem solving states. This can include reifying (explicitly displaying) the problem solving goal structure (Singley, Carroll and Alpert, 1993), or displaying implicit machine states in program execution (Eisenstadt, Price and Domingue, 1993; Ramadhan and duBoulay, 1993). Finally, the student may be asked to complete activities that are not required in the real-world problem-solving process. This can include requiring the student to specify a problem solving plan (Bonar and Cunningham, 1988; Bhuiyan, Greer and McCalla, 1992), or to specify implicit problem states (Reiser, Kimberg, Lovett and Ranney, 1992; Corbett and Anderson, 1995a). Finally, cognitive task analyses may recommend entirely reconfiguring the standard problem solving interface (Koedinger and Anderson, 1993b; Singley, 1987). We will consider these issues further in the final section on design principles.

The Domain Expert. As its name suggests, the domain expert module represents the content knowledge that the student is acquiring. This module is at the heart of an intelligent tutoring system and provides the basis for interpreting student actions. Classically, the domain module

takes the form of an expert system that can generate solutions to the same problems the student is solving.[2]

One of the very important early results in intelligent tutoring research is the importance of the cognitive fidelity of domain knowledge module: It is important for the tutor to reason about the problem in the same way that humans do. Such cognitively valid models can be contrasted with "black box" and "glass box" models (Anderson, 1988). A black box model is one that fundamentally describes problem states differently than the student. The classic example of such a system is SOPHIE I (Brown, Burton and Zdybel, 1973). It was a tutor for electronic troubleshooting that used its expert system to evaluate the measurements students were making in troubleshooting a circuit. The expert system did not apply causal human reasoning, but based its decisions by solving sets of equations. As a consequence the tutor can recommend optimal actions in each problem solving context, but it was entirely up to the student to construct a description of that problem solving context and the rationale for the appropriate action.

A glass box model is an intermediate model that reasons in terms of the same domain constructs as a human experts, but with a different control structure. The classic example of such a system is Clancey's (1982) GUIDON system that tutored medical diagnosis. This system was constructed around the expert diagnostic system MYCIN. MYCIN consists of several hundred if-then rules that probabilistically relate disease states to diagnoses. These rules reference the same symptoms and states doctors employ in reasoning, but with a radically different control structure. MYCIN reaches a diagnosis by an exhaustive back ward search. Although the expert system and doctor would agree on the description of any problem solving state, Clancey pointed out that it is difficult to tutor a doctor on what to do next when the expert system and student are following a different trajectory into and out of the state.

These studies revealed early on the likely futility of building an intelligent tutor around an off-the-shelf expert system. Instead, it is necessary to develop a true cognitive model of the domain knowledge that solves exercises in the same way students solve them. We refer to this domain expert as an ideal student model, since it is a computational model of the ideal knowledge the student is acquiring. As a result, one of the crucial steps in intelligent tutor development is a careful analysis of how humans, both expert and novice approach the problem solving task.

The Student Model. The student model is a record of the student's knowledge state. There are classically two components in a student model: an overlay of the domain expert knowledge and a bug catalog.

The first component is essentially a copy of the domain expertise model in which each knowledge unit is tagged with an estimate of how well the student has learned it. PAT, for example, assumes a simple two state learning model in which each production rule is either learned or not. As students work PAT estimates the probability that each rule is in the learned state. SHERLOCK II assumes five learning states: no knowledge, limited knowledge, unautomated knowledge, partially automated knowledge and fully developed knowledge. SHERLOCK II estimates the probabilities that each knowledge unit is in each of the five states.

[2] This applies to coaching tutors. In dialog tutors domain knowledge consists primarily of declarative knowledge. Other approaches are possible in coaching tutors; the domain knowledge module may be constructed to evaluate student solutions rather than generate solutions.

The bug catalog is set of misconceptions or incorrect rules, each carrying an indication of whether the student has acquired the misconception. Much of the early interest in intelligent tutoring systems was motivated by the promise of diagnosing and remediating student misconceptions, but this early enthusiasm has faded for two reasons. First, enduring misconceptions are less frequent than assumed (VanLehn, 1990; Sleeman, Ward, Kelly, Martinak and Moore, 1991). Van Lehn argues that many actions consistent with misconceptions are simply "patches" for problem solving impasses. While the impasse may be enduring, the student may try a different patch the next time around. Second, an influential paper by Sleeman, Kelly, Martinak, Ward and Moore (1989) indicated that providing remedial feedback tuned to bugs is no more effective than instruction that simply reteaches the correct approach. McKendree (1990) has demonstrated that carefully hand-tuned bug messages can be instructionally effective, but in general, the cost benefit analysis on bug modeling is questionable.

Statistical methods for estimating a student's knowledge state from response data has been a fundamental in psychology (cf. Green and Swets, 1973) and psychometrics (cf. Crocker and Algina, 1986). Student modeling in intelligent tutoring adds two complications. First, the student's knowledge state is not fixed, but is assumed to be increasing. We have successfully incorporated a simple Bayesian statistical model into our model tracing tutors to solve this problem (Corbett and Anderson, 1995b). This approach draws on early computer assisted learning methods (Atkinson, 1972), is employed to guide problem selection to implement mastery learning (Anderson, Conrad and Corbett, 1989; Corbett and Anderson, 1995) and is quite accurate in predicting students' posttest performance outside the tutoring environment (Corbett and Anderson, 1995b). The SHERLOCK 11 student model requires a more complex updating scheme (Lesgold, Eggan, Katz and Rao, 1992) and is used to guide problem selection and to modulate hint messages.

The second complex issue concerns grain size. Traditional psychological learning theories could assume that each response corresponds to an atomic cognitive chunk. However, student actions in complex problem solving may reflect knowledge components that can themselves be decomposed into hypothetical components. Bayesian belief nets have been introduced in intelligent tutoring systems to draw inferences about component knowledge from behavioral atoms (Gitomer, Steinberg and Mislevy, 1995; Mislevy, 1995; Martin and VanLehn, 1995).

The Pedagogical Module. The pedagogical module is responsible for structuring the instructional interventions. This module may operate at two levels (Pirolli and Greeno, 1998). At the curriculum level it can sequence topics to ensure an appropriate prerequisite structure is observed (Capell and Dannenberg, 1993) and individualize the amount of practice at each level to ensure the students master the material (Corbett and Anderson, 1995b). At the problem-solving support level, it can intervene to advise the student on problem solving activities. Towne and Munro (1992) outline five types of instructional interventions:

(I) Performance demonstration - The program demonstrates a successful sequence of actions in a problem solving task.

(2) Directed step-by-step performance- The program provides a sequence of actions for the student to follow in a problem solving task.

(3) Monitored performance - The student solves a problem set by the program. The pro gram intervenes if the student makes mistakes or gets stuck.

(4) Goal seeking - The student solves a problem set by the program. The program monitors the level of abstract problem states rather than individual actions.

(5) Free Exploration - The learner freely manipulates the problem solving environment.

A sixth condition may be included, one in which the program sets a problem solving task but does not provide support. Our model tracing tutors exemplify the third type of support, although as we shall see there are several variations on this category. SHERLOCK exemplifies the fourth category. The appropriate mix of problem solving interventions is perhaps the most contested topic in intelligent tutoring systems, in part because of the many types of learning goals that can be set. It should be noted that the educational context must be considered in making such pedagogical decisions. Optimal instructional interactions can depend on whether the intelligent tutor is working in conjunction with a classroom teacher or in isolation. We shall consider this topic further in the third section.

37.2.6 HCI Evaluation Issues

The essential measure of educational effectiveness is learning rate (as a function of cost). This can be ex pressed in two ways: Educational software environments are effective to the extent that they enable students to reach higher achievement levels in the same amount of learning time or to reach the same achievement levels with less learning time. This raises some unique issues in software evaluation. The general metric for comparing software environments is output produced divided by effort expended. In intelligent tutoring systems the observable product is problem solutions. As in any software environment, the efficiency with which the student can solve problems is an important consideration. However, the ultimate and unobservable product is knowledge. As a result, the assessment of educational software ultimately involves a transfer task: how well the students can solve problems working on their own outside the tutoring environment.

The evaluation issue is further complicated since there are multiple aspects to learning:

(1) The student needs to learn basic declarative facts in a domain, e.g., dividing the quantities on both sides of an equation maintains the equality.

(2) The student has to learn to associate these facts with problem solving goal structures. For example, If the goal is to isolate variable X in an equation of the form X*expression I = expression2, Then divide both sides of the equation by expression I.

(3) Transfer of knowledge to other environments, notably the non-tutor environment is essential.

(4) Retention (or inversely, forgetting) is an important knowledge parameter.

(5) Ideally students will acquire ‘meta cognitive’ skills in learning (Bielaczyc, Pirolli and Brown, 1995; Chi, Bassok, Lewis, Reimann and Glaser, 1989; Pirolli and Recker, 1994; Schoenfeld, 1987) to gauge their own state of knowledge and learning processes. Educational software is most successful if it supports not just learning of the specified curriculum, but helps prepare the student to ac quire subsequent knowledge.

(6) Student satisfaction or motivation or feeling of competence is also important. The student's achievement level is meaningless if the student chooses not to exercise the skill after acquiring it.

In short, the evaluation process is somewhat more complex than for most software environments. One measure of success for PAT is that the first class of PAT Algebra I students (1992-1993) were twice as likely as comparable students in traditional Algebra Ito be enrolled in academic mathematics two years later. This key piece of evaluation could only be obtained two years after students had completed Algebra I.

A final complication is that students may not be the only users. When tutors are deployed in the classroom, teachers are also users. Teachers may place many direct demands on the software: (1) entering new problems, (2) structuring the curriculum, (3) intervening in the students' current state in the system and (4) re questing summary reports. Of equal importance, a tutoring system changes the teacher's interaction with students (Schofield and Evans-Rhodes, 1990). It is important to maximize the teacher's effectiveness in these capacities.

37.3 Design and Development Methods

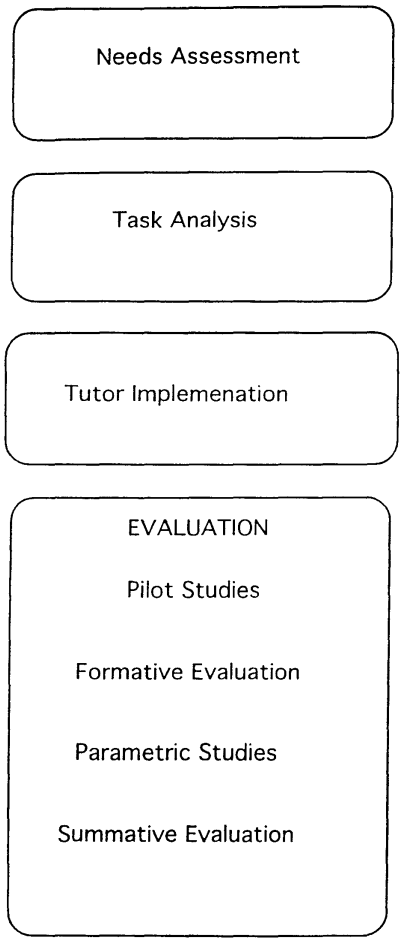

Figure 4. Intelligent tutoring design and development signs.

Figure 4 displays the major activities in tutor development: (I) needs assessment, (2) cognitive task analysis, (3) tutor implementation and (4) evaluation. The first step is common to all software design. In the case of ITS design, this involves specifying educational goals and curriculum. The second stage is common to expert systems programming, although the target is defined more narrowly here: a cognitively valid model of problem solving performance. The third phase consists of initial tutor implementation, which is followed by a series of evaluation activities: (I) pilot studies to confirm basic usability and educational impact; (2) formative evaluations of the system under development, including (3) parametric studies that examine the effectiveness of system features and finally, (4) summative evaluations of the final tutor's effect: learning rate and asymptotic achievement levels.

37.3.1 Needs Assessment

As in any software development project, the initial step is to assess user needs. This is primarily the task of the domain expert, either the expert practitioner or the teacher. As a result, educational software development is necessarily an interdisciplinary task. In the PUMP algebra project, the domain experts, both curriculum specialists and mathematics teachers identified convergent needs at the local and national levels: To include all students in academic mathematics with a reasonable chance of success, to embed academic mathematics knowledge in authentic problem solving situations so that it is relevant to both the college bound and non- college bound, and to embed it in the framework of modern computational tools. It was obvious that the goal was not to computerize traditional algebra I, but develop a new curriculum structure. It should be ob served that there is a second, pragmatic, reason to work closely with domain authorities in addition to drawing on their expertise: They share a common framework and typically common experiences with domain instructors and can play a major role in getting over the hurdle of classroom adoptions.

Having set these abstract goals, it is n to define the problem solving tasks and the role of the tutor in the curriculum (cf. Reusser, 1993). or example, in PUMP algebra students spend three class periods a week in collaborative problem solving activities and two days on individual problem solving with PAT. At CMU, on the other hand, our ACT Programming Tutor is used to teach a self-paced programming course in which students read on-line text and complete programming exercises on their own. It is an equally important step to decide what will count as success. We strongly believe that this should be defined in terms Of student, and to a lesser extent teacher behavior; and should be defined before tutor development begins. The critical measure is not the coverage of the tutor in interpreting student behavior. It is also not an abstract sense of what the student knows, but rather an objective measure of what the student can do. There are per haps three crucial dimensions: (I) the probability a student is able to solve problems; (2) the time it takes to reach this performance level and (3) the probability the student will actively use this knowledge in the future. Secondarily, it should be noted that educational software will only be successful to the extent that vendors are willing to sell it, administrators to buy it and teachers to use it.

This focus on an operational definition of knowledge goes along way to defining a curriculum, since the best way to acquire a skill is to practice performing the skill. In the case of the PUMP algebra project, the skill of manipulating symbolic expressions was de emphasized and the skill of modeling authentic problem situations in various formalisms was emphasized. This led directly to a curriculum specification and classroom activities. The tutor in turn was integrated in the overall structure of the course from its inception.

Along with curriculum goals, the technology base must be assessed. A limiting factor historically in the penetration of intelligent tutors into the real world has been the cost of artificial intelligence workstations. When we first piloted the Geometry Proof Tutor (GPT) in a city high school ten years ago, each workstation cost $20,000. Technology costs have declined more than 8-fold since then and a workstation in our PUMP algebra labs costs under $2500. The cost of providing a workstation to every student in a classroom, about $60000, is well within reach of most institutions, but is not a discretionary spending amount. Funding for high school labs has been the greatest barrier to widespread dissemination.

Student entry characteristics must also be assessed. For example, many students entering PUMP algebra have not mastered middle school mathematics. Consequently learning experiences with the earliest versions of PAT did not generalize in the expected way. Al though in principle students have acquired the unifying concept of rational number in middle school, students who successfully manipulate expressions with positive integer coefficients and terms in PAT often struggle with negative integer and fractional components. While entry characteristics may be predicted by experts, in the end they must be determined empirically. There are two important points to make here. First, while needs assessment is logically prerequisite to software design,

it is an iterative process. We only recognized students' entry characteristics after they started working with the system. Second, the mismatch between student entry characteristics and the tutoring system was not revealed by breakdowns in our Al representation of do main knowledge. The tutor was perfectly capable of coping with each student action irrespective of the students' various entry characteristics. Instead, it was revealed by detailed monitoring of transfer of student performance across contexts.

Finally, teacher entry characteristics must be assessed. It should be noted that teachers as well as students are system users when tutors are deployed in conventional educational settings and the technology it self can be a challenge for teachers. It is important to provide sufficient in-service training for teachers to feel comfortable with the technology. Currently, new teachers in the PUMP project engage in a week of full-time in-service training just before the school year starts.

In summary, for any educational technology to succeed it must be used. To be used it must be integrated with the other classroom activities and must be accepted by the teachers and it must be appropriate for students. These are among the goals of the initial needs assessment.

37.3.2 Cognitive Task Analysis

Following the needs assessment, the cognitive scientist enters the picture. The goal of the cognitive task analysis is to develop a psychologically valid computational model of the problem solving knowledge the student is acquiring. There are three chief methods for developing a cognitive model: (I) interviewing domain experts, (2) conducting "think aloud" protocol studies with domain experts and (3) conducting "think aloud" studies with novices. The first approach is the most common and easiest approach. It includes a common situation in which the cognitive scientist is familiar with the field in question and simply generates the cognitive model. However, it is generally recognized that interviewing experts, including the cognitive scientist, is not a sufficient method to develop a pedagogically optimal model. An important goal of the task analysis is to recognize the expert's implicit goal structure (Koedinger and Anderson, I 993b), heuristics and mental models (Means and Gott, 1988; Kurland and Tenney, 1988), and experts

often prove incapable of reporting these cognitive components. The preferred method to obtain cognitive model evidence is the think aloud study (Ericsson and Simon, 1984) in which the expert is asked to report aloud what she is thinking about in solving typical problems.

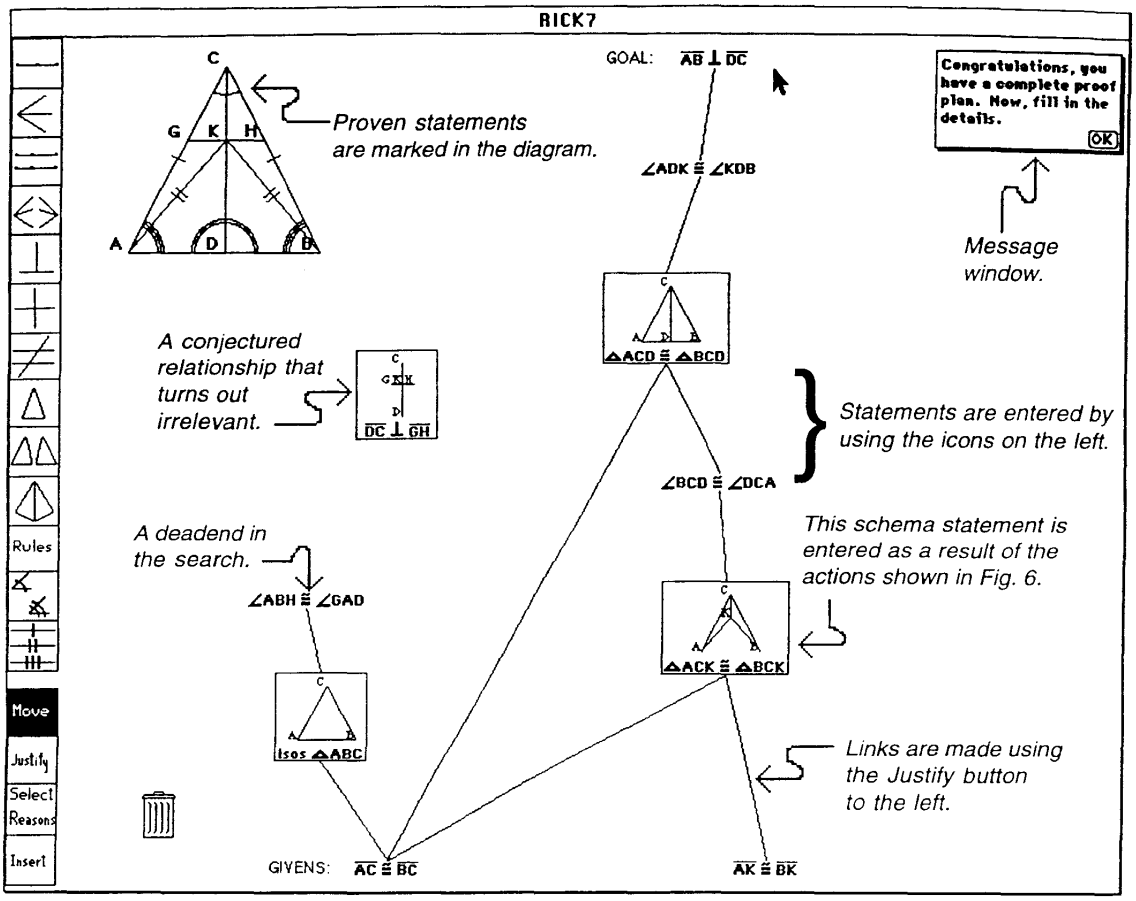

Figure 5. The ANGLE geometry tour.

For example, think aloud protocols of geometry teachers working proofs (Koedinger and Anderson, 1990) revealed that these experts departed substantially from the expected mechanisms of step-by-step forward chaining from problem givens and backward chaining from the goal statement to be proved. Instead, these experts begin by inspecting the problem diagram, recognizing familiar visual patterns and reasoning about the conclusions that can be proved independent of the problem solving goal. Only then do they consider the goal itself. The second major difference is that these experts reason through their proof by setting major subgoals and then go back and fill in details. This led to a geometry proof tutor ANGLE that (a) employs visual icons that correspond to the experts' perceptual processing of diagrams and that (b) allows students to post subgoals. Figure 5 displays the ANGLE interface. In this r environment students build a proof graph (Anderson, Boyle and Reiser, 1985) rather than a two column proof. Students construct each statement in the proof by selecting an icon from the set of icons on the left side of the screen that reify the types of visual patterns the experts detect. As can be seen in the figure, students can post free floating sub goals in this interface, as well as working directly for ward from the givens or backward from the goal.

Finally, observation of novice problem solvers is desirable to identify aspects of problem solving that students find difficult (Anderson, Farrell and Sauers, 1984; Means and Gott, 1988; Kurland and Tenney, 1988). At minimum such observations will reveal unwarranted assumptions. A few issues emerged from the PUMP curriculum. For example, in labeling spread sheet columns and in labeling graph axes it turned out that students blurred the distinction between the quantity being measured, e.g., time, and the unit measurement, e.g., hours. This led us to revise not just the cognitive model for PAT, but to alter the problem solving task and interface to incorporate labeling activities that we took for granted. It should be noted that if novice problem solving studies are not conducted in this phase, they are necessarily completed in subsequent tutor evaluation phases.

37.3.3 Tutor Implementation

Following the cognitive task analysis, we are ready to implement the tutor. As described earlier, the goal is to set up a problem solving environment that enables authentic problem solving and that helps support the problem solving process. The cognitive analysis yields an expert system that can reason about the problems, but may also yield interface modifications as in the case of ANGLE. We will postpone a consideration of tutor design principles to the third section of this chapter. Two points are worth noting. First, we estimate that 200 hours of intelligent tutor development are required to deliver a full hour of instruction. Second, evaluation should begin as early as possible in the development process.

37.3.4 Evaluation

The phases in educational software evaluation are similar to that in any software evaluation. Pilot studies follow quickly on the heals of tutor development as a rapid check on the usability and effectiveness of the tutor. Formative evaluation and revision iterate through the development of the system. The development process is capped by summative evaluations of the educational impact of the tutor, ideally in the form of field testing.

Piloting. Initial pilot studies are intended to reveal serious misjudgments concerning the problem solving environment and to establish that the tutor supports a minimally acceptable degree of learning. In a typical study, students complete a pretest, work through problems with the tutor and complete a post-test. Pilot studies with PAT revealed an unexpected difficulty in graphing. While the cognitive model focuses on issues in point and line plotting, it emerged that students did not have good heuristics for setting the axis scales and bounds dependent on the size of the values being plotted in each problem. Students performed these activities in the interface, but we did not model them in the domain expert. Consequently the tutor was unable to help as students floundered in setting up their graphs. We subsequently expanded the cognitive model to Support this activity.

Formative Evaluations. Formative evaluations can focus on any of the four intelligent tutor components. In our lab, our analyses primarily focus on the validity of the cognitive model with respect to educational out comes. In particular, we seek evidence that the students are not learning the ideal rules in the cognitive model, as revealed in failure of the knowledge to generalize across expected contexts. For example, if we trace out successive opportunities to apply production rules, we typically find regular learning curves. That is, error rate and response time in applying a cognitive rule tend to decline systematically across opportunities to apply the rule, as shown in Figure 6.

Figure 7, in contrast, displays the learning curve for one rule in the original cognitive model of the APT Lisp Programming Tutor:

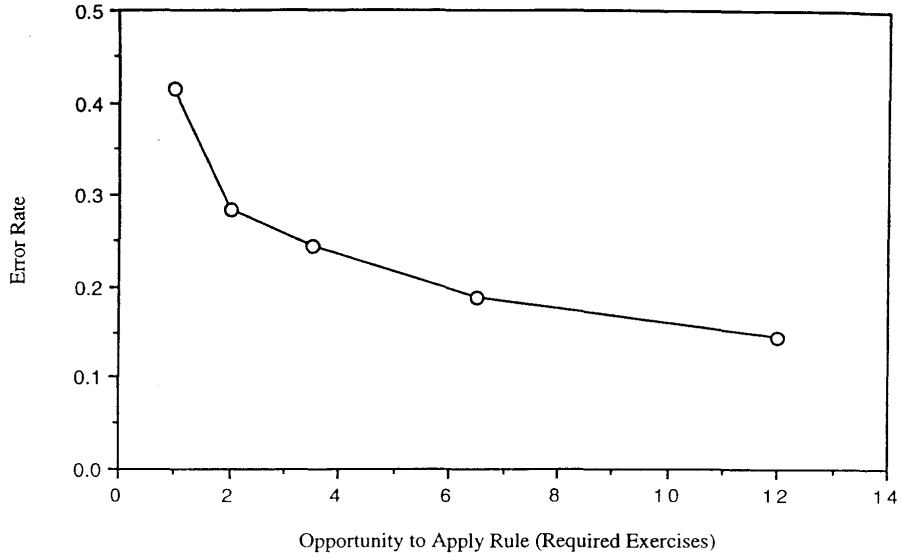

Figure 6. The mean learning curve for coding rules introduced in the first lesson of the Lisp tutor curriculum. *Mean error rate is plotted for successive opportunities to apply the rules. From A. Corbett and J. Anderson 1995b. Copyright 1995 by Kluwer Academic Publishers. Reprinted by permission.*

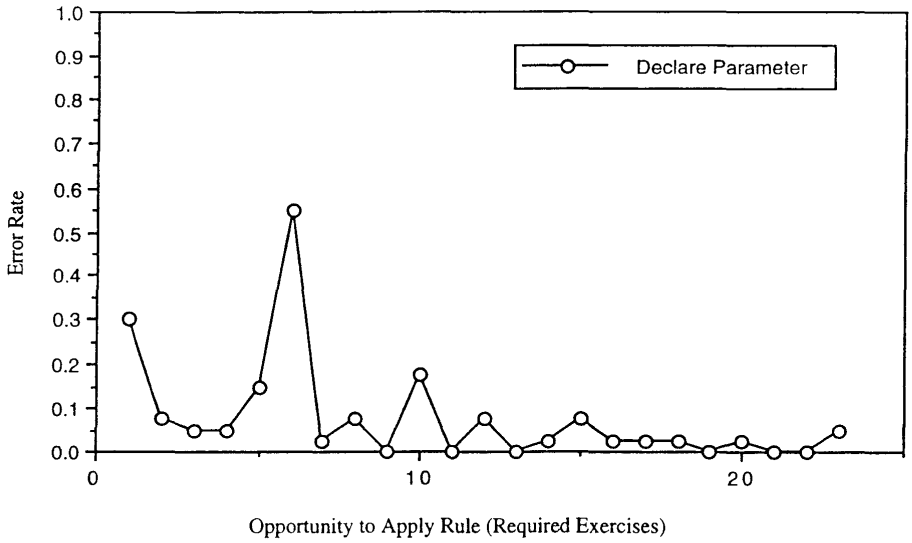

Figure 7. The learning curve for a hypothetical rule that declares a variable in a function definition. *Error rate is plotted for successive opportunities to apply the rule. From A. Corbett and J. Anderson 1995b. Copyright 1995 by Kluwer Academic Publishers. Reprinted by permission.*

In writing a Lisp program, declare one parameter variable for each argument (input value) that will be passed when the program is executed.

This rule is expected to fire each time a variable is declared in a function definition. As can be seen, error rate for this rule decreases monotonically over the first four applications, but then jumps for the fifth and sixth applications. The first four points are drawn from the first four programming exercises, in which a single variable is declared. The fifth and sixth points represent the fifth programming exercise and the first definition in which two parameter variables are declared. The seventh and eighth points represent two more function definitions with a single parameter and the ninth and tenth points represent the next exercise that requires multiple parameters. The 55% error rate at goal six in this figure suggests that just under half the students had encoded the ideal rule initially and were able to apply it in the new situation. The other students initially encoded a non-ideal rule that was sufficient to complete the first four exercises but failed to generalize. There are several possible non-ideal rules:

> Always code one variable to stand for the whole list of arguments (too general).
>
> Code one variable when there is one argument (too specific - no thought of more arguments).
>
> Code one variable because that's what the ex ample has (no functional understanding).

Similar patterns were observed for a variety of ideal rules in the model.

We revised the Lisp Tutor cognitive model in response to this problem so that procedural knowledge for variable declarations is represented as two rules:

> In defining a Lisp function, declare one variable for the first argument that will be passed.
>
> In defining a Lisp function, declare one additional variable for each additional argument that will be passed.

Figure 8 re-plots the data points for declaring variables. The points from the original learning curve in Figure 7 are plotted as two separate learning curves for these rules. Under this analysis the sixth, tenth, twelfth, fifteenth, seventeenth, nineteenth and twenty-third opportunities to declare a variable are really the first through seventh opportunities to apply the second rule.

In the course of successive refinements of this part of the Lisp Tutor curriculum the cognitive rule set more than doubled. While there is a natural tendency to write general rules in the ideal student model that apply in multiple contexts, there are two reasons to accurately represent the actual though perhaps less-general rules students are encoding. One is to accurately represent the student's knowledge state for purposes of judging mastery as knowledge tracing does. The second is to provide context-specific help messages in the psychologically distinct contexts.

Parametric Evaluations. In addition to naturalistic observations of students and teachers working with the tutor, formative evaluations can include experimental manipulations of tutor components. In intelligent tutoring systems these parametric evaluations have tended to focus on pedagogical issues, notably feedback con tent and control, as described in the third section on design issues (Corbett and Anderson, 1991; Corbett, Anderson and Patterson, 1990; Koedinger and Anderson, in press; Reiser, Copen, Ranney, Hamid and Kim- berg 1995; Shute,

1993). These studies focus on the impact of the manipulation on learning time, asymptotic achievement levels, affective measures and individual differences.

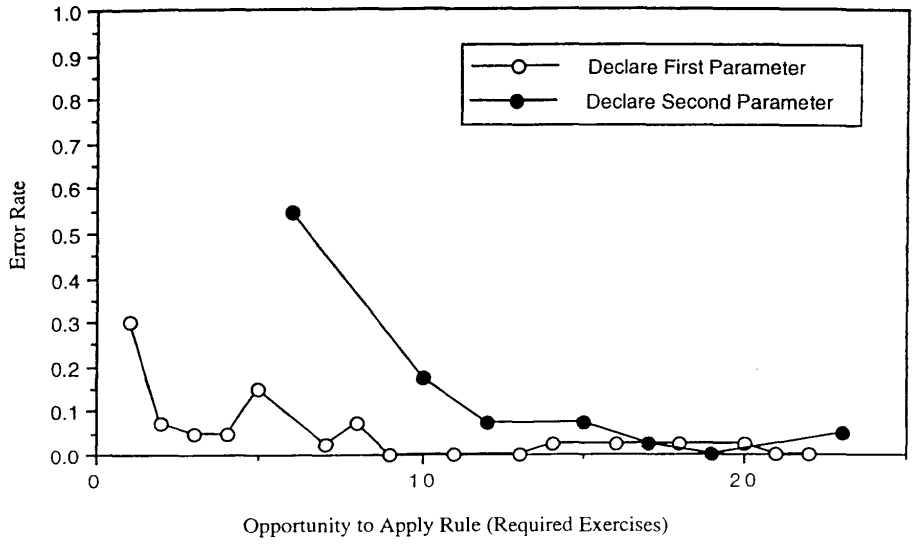

Figure 8. A partitioning of the single learning curve for variable declarations in Figure 7 into separate learning curves for two distinct variable declaration rules. *From A. Corbett and J. Anderson 1995b. Copyright 1995 by Kluwer Academic Publishers. Reprinted by permission.*

Summative Evaluation. Summative evaluations examine the overall educational impact of the tutoring sys tem. Such evaluations compare the intelligent tutoring environment to standard educational conditions. Ideally, such summative evaluations take the form of field tests in which the tutor is deployed in an authentic educational environment. Necessarily such studies compare learning outcomes — assessing students' problem solving success in post tests following learning. Pre tests of problem solving ability are not strictly necessary in such comparison across learning environments, although they are important if students are assigned non-randomly to condition. A difficult issue in such summative evaluations is that the natural comparison conditions may have different educational goals. For example, the educational goals of traditional algebra I differ from the goals of PUMP algebra. Our solution to this problem is to present posttests appropriate to each learning environment to all students (Koedinger, Anderson, Hadley and Mark, 1993). In the case of our tutoring project, students in the PUMP algebra classes were 100% better at reasoning among multiple representations in problem solving, but also 10-15% better on standardized algebra tests targeted by traditional algebra classes.

Learning time is the second principal assessment measure in summative evaluations. Intelligent tutors make an important contribution to education when they speed learning substantially. This can happen in two ways. First, they can increase the rate at which practice opportunities are presented. For example, SHERLOCK offers what would ordinarily be several years' worth of troubleshooting opportunities in the space of several days of practice. Second, tutors can increase the rate at which students learn from practice opportunities. For example, Corbett and Anderson (1991) found that students working with the APT LISP Tutor completed a fixed set of programming exercises in one third the time required by students working in the same editing interface without tutorial support and performed reliably better on a posttest.

37.4 Design and Development Principles

In this section we consider a set of general principles for intelligent tutor design and conclude with a discussion of pedagogical guidelines concerning help messages. In 1987 we published a set of eight principles for intelligent tutor design (Anderson, Boyle, Farrell and Reiser, 1987). Recently we revisited those principles and found them generally sound (Anderson, Corbett, Koedinger and Pelletier, 1995). However, we have come to believe that a single overarching principle governs intelligent tutor design:

Principle 0: Enable the student to work to the successful conclusion of problem solving.

While there are many variations on intelligent tutoring environments, we believe that the major factor in their success comes from enabling the student to reach a solution to each problem. (cf. Lesgold, Eggan, Katz and Rao, 1992). Our strongest evidence for this principle comes from a study with the APT Lisp Tutor. The Lisp Tutor is a model tracing tutor that assists students is completing short Lisp programming problems. We employed this tutor in a study of feedback timing and control (Corbett and Anderson, 1991) with four feedback conditions. One was a standard immediate feedback condition in which the tutor provides feed back on each symbol. A second, error flagging condition, flagged errors immediately, but did not require correction. Students could correct errors when they chose. The tutor also checked code by testing so it was possible for students to succeed with a program definition the tutor did not recognize. In a third condition, feedback-on-demand, the tutor provided no assistance until requested by the student. When asked for assistance in this condition, the tutor traces through the student's code symbol-by-symbol, and reports the first error it finds. In the fourth condition, students completed programming exercises in the same problem solving environment, but without any tutorial assistance. Students in the three feedback conditions were required to reach a correct solution to each problem, since the environment provided sufficient feedback to guarantee success. In the control condition, students could work on an exercise for as long as they desired, but in the end were allowed to give up, in which case the tutor displayed a correct solution. There were substantial differences in the time taken to complete the fixed set of exercises, but more importantly, here, students in the baseline control failed to solve approximately 25% of the exercises. Students completed posttest in three different environments, both on-line and paper-and-pencil. There were no reliable accuracy differences across the three posttest environments among the three feedback groups, but the feedback groups performed reliably better than the control group in all posttest environments.

37.4.1 Eight Design Principles

As indicated above, we believe the following eight de sign principles have stood the test of time:

(1) Represent student competence as a production set. This represents a theoretical stance on a more general principle: represent problem solving knowledge IS a psychologically valid way. A substantial body of empirical evidence supports the validity of a production rule decomposition of procedural knowledge (Anderson, Conrad and Corbett, 1989), but it should be noted that alternative representations have been employed in intelligent tutoring systems, including plan based representations (Johnson and Soloway, 1984) and case-based representations (Riesbeck and Schank, 1991; Weber, 1993, in press).

(2) Communicate the goal structure underlying the problem solving. Successful problem solving involves the decomposition of the initial problem state into sub goals and bringing domain

knowledge to bear on those subgoals. This is a universally accepted principle. SHERLOCK is a strong embodiment of the principle; it explicitly models problem solving goal states, but not specific action sequences to achieve the goals. Again, there are two general methods for communicating the goal structure. First, it can be reified in the problem solving environment. For example, the problem solving goal tree can be explicitly represented on the computer screen (Singley, Carroll and Alpert, 1993) or the problem solving formalism can be modified to reflect the problem subgoal structure, as it is in the ANGLE's proof tree representation (Koedinger and Anderson, 1993a). Second, the goal structure can be communicated through help messages. For example, when a student asks for help in our model tracing tutors, the first level of help at each problem solving step is a reminder of the current goal in the context of the overall problem (the current state of the problem and the desired sub goal state). Subsequent help messages advice on how to accomplish the goal.

(3) Provide instruction in the problem solving context. As suggested earlier, this is an almost universally accepted principle that cuts across disparate educational philosophies ranging from behaviorism to Constructivism.

(4) Promote an abstract understanding of the problem-solving knowledge. Again, this is a well-accepted principle. Means and Gott (1988) state: "The most powerful aspect of expertise, and at the same time, the hardest aspect to capture is what Sleeman and Brown (1982) called its 'abstracted' character." The goal is to help students to encode problem solving states, actions and consequences in the most general form consistent with the problem solving goal in order to foster transfer across contexts. Following the formative evaluation of SHERLOCK, the design team began analyzing 'families' of tasks over which transfer was expected (Lesgold, Eggan, Katz and Rao, 1992). A hierarchy of general methods and task-specific specialization was defined that enabled SHERLOCK II to provide feedback at an appropriate level of abstraction. Hint messages in our model tracing tutors are also constructed to provide an abstract description of how to achieve the current problem solving goals. A concrete action is described only as a last resort when there is no expectation that a teacher will be present in the environment to interact with the student.

(5) Minimize working memory load. Sweller (1988) demonstrated that a high working memory load can interfere with problem solving. There are multiple tactics to reduce working memory load in an intelligent tutoring system. Efforts described earlier to reify problem solving states and device states in the problem solving interface serve to reduce working memory load as well as to foster comprehension. The interface may also simplify problem solving actions to reduce the load on working memory as well as allowing students to focus on a more abstract problem solving level. For example, our programming tutors employ structure editors that remove much of the burden of remembering surface syntax. SHERLOCK simplifies the task of taking measurements to mouse clicks. Littman (1991) observes behavior by human tutors that is consistent with this principle; they do not interrupt students (and their current working memory state) to point out relatively minor errors that have little consequence for the student's overall goal structure. As Lesgold, Eggan, Katz and Rao (1992) observe there can be a conflict between principles (3) and (5), since setting a problem solving goal imposes a working memory load that can interfere with full processing of the problem solving state. The authors propose a reflective review at the conclusion of problem solving as a solution to this conflict in this reflective review process students replay there solutions step-by-step. They can examine the device state and problem solving state at each step and also get advice on how an expert would solve the problem.

(6) Provide immediate feedback on errors. This has been our most controversial principle. The chief benefit of immediate feedback is to reduce learning time substantially (Anderson, Corbett, Koedinger and Pelletier, 1995). However, others point out that immediate feedback can interfere with aspects of learning. We discuss this issue in more detail under pedagogical guidelines below.

(7) Adjust the grain size of instruction with learning. As students practice and become familiar with standard goal decompositions in problem solving tasks, they should be able to reason in terms of relatively high level goals and instruction should be targeted at these high level goals. For example, consider the equation 2X + 4 8. Early in equation solving practice, instruction may focus on the individual steps of collecting constant terms and dividing by 2 (or vice versa). Once these tactics have been mastered, it should no longer be necessary to interact pedagogically on the individual steps in achieving these goals. Similarly in programming, once a student has learned well-defined algorithms such as reading and totaling a sequence of inputs, it may be necessary in later to instruction to advise that the algorithm is called for, but it should not be necessary to support the student in writing the appropriate code. Little empirical work has been directed at this principle.

(8) Facilitate successive approximations to the target skill. This principle suggests that a student's problem solving performance should gradually come to reflect real-world problem solving performance. In one sense this happens automatically; as the students practices and becomes more skilled the tutor is called upon to provide less assistance. A second important implication of this principle is that scaffolding in the problem solving environment should fade as the student continues practicing.

37.4.2 Pedagogical Guidelines

In this section we consider guidelines for providing help in intelligent tutoring systems. When to present advice and in what form has been an enduring issue in the field, in large part because of the complexity of the learning process. At one end of the continuum, some environments based on device or system simulations emphasize student exploration and discovery learning (Shute and Glaser, 1990). At the other end, some environments provide immediate feedback on each student action (Anderson, Corbett, Koedinger and Pelletier, 1995). Other environments provide assistance only when asked by the student or when the student seems to be in serious trouble (Corbett, Anderson and Patter son, 1990; Lesgold, Eggan, Katz and Rao, 1992). Meta-analyses of feedback timing reveal little consistent impact on the acquisition of target skills (Bangert Drowns, Kulik, Kulik and Morgan, 1991; Kulhavy, 1977; Kulik and Kulik, 1988). However, feedback timing can have indirect effects on the acquisition of a complex skill. Immediate feedback can effectively speed the acquisition of a target skill, by reducing floundering. However, immediate feedback can interfere with the acquisition of associated skills.

One approach to this issue has been to examine what human tutors do (Merrill, Reiser, Ranney and Trafton, 1992). In general, human tutors do provide immediate feedback, if only on "important" errors (Littman, 1991). Two studies (Lepper, Aspinwall, Mumme and Chabay, 1990; Fox 1991) indicate that human tutors effectively provide feedback on each action, perhaps just by saying "ok" They signal errors subtly, perhaps by pausing before saying 'ok', allowing students ample opportunity to correct errors on their own and giving students a sense of control. We have converged on this type of feedback in our model tracing tutors, particularly in view of Sleeman et. a1.'s (1989) results on the ineffectiveness of diagnostic feedback. In our early tutors we provided immediate diagnostic feedback on errors, when possible (about 80% of the

time). Our current generation of mathematics and programming tutors still conventionally pro vides immediate feedback on each problem solving action, but does not provide advice on the nature of the error or how to arrive at a correct action. This prevents the student from wasting time going down the wrong path, but allows the student to rethink the problem solving situation, much as a human tutor does.

There is a second important point to make here:

While intelligent tutoring systems are modeled on human tutors, it is not clear that the analogy should be taken too literally. Indeed, we have moved away from the human tutor analogy for two reasons. First, it is too high a standard to live up to; if you set the standard of a human tutor, the user is going to be sorely disappointed. Second, we want students and teachers to view these tutors as learning tools. That is, we want students to think of the tutors as tools they are employing, rather than as taskmasters. Similarly, we want teachers to think of the tutors as tools that can free their time to interact individually with students. Schofield, Evans-Rhodes and Huber (1990) studied the impact of our original geometry tutor GPT in the class room and found two relevant results, First, although it provided immediate feedback it was highly motivating for students and teachers. Second, students came to perceive their teacher as a collaborator in learning rather than as a remote expert. In short, the overall effect of the tutor in the classroom was to give students a greater sense of control.

We propose the following guidelines for feedback timing:

(1) feedback should be provided at a time when the relevant information (problem state and action consequences) can be communicated effectively to the student,

(2) feedback should not change the requirements of the task,

(3) feedback should not disrupt performance of the task,

(4) to optimize learning of the target skill, as measured in elapsed time rather than in elapsed problem solving exercises, feedback should be presented as early as possible.

Results of a variety of past studies on feedback timing in skill acquisition have been consistent with these principles. For example, Schmidt, Young, Swinnen and Shapiro (1989) examined the acquisition of a motor task requiring a small set of precise movements with no environmental feedback. While students in an immediate feedback condition appeared to acquire the task more quickly than students in a blocked feedback condition, they performed worse in no-feedback post- tests. In this case, immediate feedback after each movement substantially changes the task, however. Performance in a no-feedback environment requires a long-term internal model of the motion. Students in the immediate feedback condition, in contrast, may succeed by making adjustments in a transient trace of the immediately preceding performance. Munro, Fehling and Towne (1985) found immediate feedback to be detrimental in a radar monitoring task, but the feedback interfered with the real-time demands of the task. Lewis and Anderson (1985) found that immediate feedback was superior to delayed feedback, but this was in an adventure task in which it was difficult to reinstate past problem states. Corbett and Anderson (1991; Corbett, Anderson and Patterson, 1990) found it took students substantially longer to complete a fixed set of programming exercises with delayed feedback than with immediate feedback, although there were no differences in posttest accuracy levels across groups.

As suggested above, immediate feedback on a tar get skill can interfere with acquisition of other problem solving skills. Classically, in the case of programming, immediate feedback on program generation blocks practice on program debugging. In general, immediate feedback on solution generation interferes with the development of meta-cognitive self monitoring skills and error recovery skills. For example, Reiser, Copen, Ranney, Hamid, and Kimberg (1995) found that students who learned to write programs in an exploratory environment developed greater debugging skills than students who learned to write programs with immediate feedback. This effect interacted with student back ground however. Students with strong mathematics and programming backgrounds acquired greater debugging skills in the discovery environment, while students with weak backgrounds did not. We believe that this result highlights the fact that this is a curriculum issue as much as, or more than a pedagogical issue. Ideally, an intelligent tutoring environment should be based on a cognitive task analysis of all aspects of problem solving, including solution generation, error recovery and self-monitoring, and should support the student in all aspects of problem solving.

A general consensus seems to be emerging on the content of advice messages, along the following lines. When error feedback is presented, it should generally just signal the error without commenting. This enables the student maximum opportunity to analyze the correct the situation. When advice is given, the most cost effective content focuses on “reteaching” a correct analysis of the situation rather than debugging misconceptions. This correct analysis should be administered in at least three or four stages: (I) a reminder of the problem solving goal, (2) a description of relevant features of the current problem state and the desired goal state, (3) a description of the rule for moving from the current state to the desired state, and (4) a description of a concrete action to take.

Shute (1993) provides evidence of a second “aptitude-treatment” interaction with implications for the content of help messages. In this study students were learning some basic laws of electricity in an intelligent tutoring system. Students received feedback on their solution to each problem and Shute developed two variations on the tutoring environment with respect to this feedback. In the rule application version, the tutor described an applicable rule, i.e., the key relation ship among a set of variables. In the rule induction version, the tutor pointed out the relevant variables but left it up to the student to induce the relevant relation ship. Shute measured students exploratory behavior and found that high-exploratory students were more successful in the induction environment, while low exploratory students were more successful in the more directive application environment. This suggests that student learning style as well as current knowledge state should govern the content of tutor advice. Notice, however, that the graduated advice framework de scribed in the preceding paragraph may accomplish the same goal without the need for student modeling. Shute’s high-exploratory students benefit most from level (2) advice while the low-exploratory students benefit most from level (3) advice. It would be an. interesting study in meta-cognitive skills to see if students zero in on the level of help they find most useful in such an environment.

37.5 Conclusion

Intelligent tutoring systems have provided a fertile ground for artificial intelligence research over the past twenty-five years. Some of these systems have been demonstrated to have a very large impact on educational outcomes in field tests, including effective learning rate, asymptotic learning levels and motivation. We are just entering a time when intelligent tutoring systems can have a real impact in the educational marketplace as technology costs decline, but we believe this will happen only if ITS research focuses on educational outcomes as well as Al issues. In

this paper we describe a process model for developing effective intelligent tutoring systems that includes a needs assessment, cognitive task analysis and early and extensive evaluation. We also describe what we view as an emerging consensus on principles and guidelines for ITS design, but recognize that more formative evaluation is required to assess and refine these prescriptions. New issues continue to emerge in the area, including ITS authoring environments, collaborative learning, world wide web deployment and virtual reality environments. However, it is important for the field to re main focused on valid pedagogical principles and educational outcomes in exploring these areas.

37.6 References

Anderson, J.R. and Bower, G.H. (1973). Human associative memory. Washington, DC: V.H. Winston and Sons.

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

Anderson, J.R. (1988). The expert module. In M. Polson and J. Richardson (Eds.) Foundations of intelligent tutoring systems. Hillsdale, NJ: Lawrence Erlbaum Associates.

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

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

Anderson, J.R,, Bo C.F. and Reiser, B.J. (1985). Intelligent tutoring systems. Science, 228, 456-462.

Anderson, J.R., Conrad, F.G. and Corbett, A.T. (l98 Skill acquisition and the LISP Tutor. Cognitive Science, 13, 467-505.

Anderson, J.R. and Corbett, A.T. (1993). Tutoring of cognitive skill. In J. Anderson (Ed.) Rules of the mind. Hillsdale, NJ: Lawrence Erlbaum Associates.

Anderson, J.R., Corbett, A.T., Koedinger, KR. And Pelletier, R. (1995). Cognitive tutors: Lessons learned. The Journal of the Learning Sciences, 4, 167-207.

Anderson, J.R., Farrell, R. and Sauers, R. (1984). Learning to program in LISP. Cognitive Science, 8, 87-129.

Anderson, J.R. and Reiser, B.J. (1985) The LISP Tutor. Byte, 10, 159-175.

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

Bangert-Drowns, R.L., Kulik, C.C., Kulik, J.A and Morgan, M. (1991). The instructional effect of feed back in test-like events. Review of Educational Re search, 61,213-238.

Bielaczyc, K., Pirolli, P.L. and Brown, A.L. (1995). Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. In Cognition and Instruction, 13, 22 1-252.

Bhuiyan, S., Greer, J.E. and McCalla, G.I. (1992). Learning recursion through the use of a mental model- based programming environment. In C. Frasson, G. Gauthier, G. and G. McCalla, (Eds.) Intelligent tutoring systems: Second International Conference, ITS '92. New York: Springer-Verlag.

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.

Bonar, J.G. and Cunningham, R. (1988). Bridge: Tutoring the programming process. In J. Psotka, L. Massey and S. Mutter (Eds.) Intelligent Tutoring Systems: Les sons learned. Hillsdale, NJ: Lawrence Erlbaum Associates.

Brown, J.S., Burton, R. and Zdybel, F. (1973). A model-driven question-answering system for mixed initiative computer-assisted instruction. IEEE Tran.. actions on Systems, Man and Cybernetics. SMC3, 248-257.

Brusilovsky, P. (1995). Intelligent learning environments for programming: The case for integration and adaptation. Proceedings of the 7th World Conference on Artificial Intelligence in Education.

Burton, R.R. and Brown, J.S. (1982). An investigation of computer coaching for information learning activities. In D. Sleeman and J. Brown (Eds.) Intelligent tutoring systems. New York: Academic Press.

Capell, P. and Dannenberg, R.B. (1993). Instructional design and intelligent tutoring: Theory and the precision of design. Journal of Artificial Intelligence in Education, 4,95-121.

Carbonell (1970). Al in CA!: An artificial intelligence approach to computer-assisted instruction. IEEE Transactions on Man-Machine Systems, 11,190-202.

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

Clancey, W.J. (1982). Tutoring rules for guiding a case method dialogue. In D. Sleeman and J. Brown (Eds.) Intelligent tutoring systems. New York: Academic Press.

Cohen, P.A., Kulik, J.A. and Kulik, C.C. (1982). Educational outcomes of tutoring: A meta-analysis of findings. American Educational Research Journal, 19, 237-248.

Collins, A.M. and Loftus, E.F. (1975). A spreading- activation theory of semantic processing. Psychology cal Review, 82, 407-428.

Collins, A.M. and Stevens, A.L. (1991). A cognitive theory of inquiry teaching. In P. Goodyear (Ed.) Teaching knowledge and intelligent tutoring. Nor- wood, NJ: Ablex Publishing.

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

Corbett, A.T. and Anderson, J.R. (1995a). Knowledge decomposition and subgoal reification in the ACT Programming Tutor. Proceedings of the 7th World Conference on Artificial Intelligence in Education..

Corbett, A.T. and Anderson, J.R. (1995b). Knowledge 'racing: Modeling the acquisition of procedural knowledge. User modeling and user-adapted interactions 4, 253-278.

Corbett, AT., Anderson, J.R. and Patterson, E.G. (1990). Student modeling and tutoring flexibility in the Lisp Intelligent Tutoring System. In C. Frasson and G. Gauthier (Eds.) Intelligent tutoring systems: At the crossroads of artificial intelligence and education. Norwood, NJ; Ablex Publishing.

Crocker, L. and Algina, J. (1986). Introduction to classical and modern test theory. New York: Harcourt Brace Jovanovich College Publishers.

Eisenstadt, M., Price, B.A. and Domingue, J. (1993). Redressing ITS fallacies via software visualization. In E. Lemut, B. du Boulay and G. Dettori (Eds.) Cognitive models and intelligent environments for learning programming. New York: Springer-Verlag.

Ericsson, K.A. and Simon, H.A. (1984). Protocol analysis: Verbal reports as data. Cambridge, MA: The MIT Press.

Fox, B. (1991). Cognitive and interactional aspects of correction in tutoring. In P. Goodyear (Ed.) Teaching Knowledge and Intelligent Tutoring. Norwood, NJ: Ablex Publishing.

Frasson, C., Gauthier, G. and McCalla, G.I. (1992).

Intelligent tutoring systems. Second International Conference, ITS'92. New York: Springer-Verlag.

Frasson, C., Gauthier, G. and Lesgold, A. (1996). Intelligent tutoring systems: Third International Conference, ITS'96. New York: Springer-Verlag.

Gitomer, D.H., Steinberg, L.S. and Mislevy, R.J. (1995). Diagnostic assessment of troubleshooting skill in an intelligent tutoring system. In P. Nichols, S. Chipman and R. Brennan (Eds.) Cognitively Diagnostic Assessment. Hilisdale, NJ: Lawrence Erlbaum Associates.

Green, D.M. and Swets, J.A. (1973). Signal Detection theory and Psychophysics. Huntington, NY: Robert E. Krieger Publishing.

Johnson, W.L. and Soloway, E.M. (1984). PROUST: Knowledge-based program debugging. Proceedings of the Seventh International Software Engineering Conference, Orlando, FL.

Katz, S. and Lesgold, A. (1993). The role of the tutor in computer-based collaborative learning situations. In S. Lajoie and S. Derry (Eds.) Computers and Cognitive Tools. Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Koedinger, K.R. and Anderson, J.R. (1990). Abstract planning and perceptual chunks: Elements of expertise in geometry. Cognitive Science, 14, 5 11-550. Koedinger, K.R. and Anderson, J.R. (1993a). Effective use of intelligent software in high school math class rooms. In P. Brna, S. Ohisson and H. Pain (Eds.) Proceedings of AJED 93 World Conference on Artificial Intelligence in Education.

Koedinger, K.R. and Anderson, J.R. (1993b). Reifying implicit planning in geometry: Guidelines for model- based intelligent tutoring system design. In S. Lajoie and S. Derry (Eds.) Computers and Cognitive Tools. Hilisdale, NJ: Lawrence Eribaum Associates.

Koedinger, K.R. and Anderson, J. R. (in press). Illustrating principled design: The early evolution of a cognitive tutor for algebra symbolization. Interactive Learning Environments.

Koedinger, K.R., Anderson, J.R., Hadley, W.H. and Mark, M.A. (1995). Intelligent tutoring goes to school in the big city. Proceedings of the 7th World Conference on Artificial Intelligence in Education.

Kulhavy, R.W. (1977). Feedback in written instruction. Review of Educational Research, 47, 211-232.

Kulik, J.A. and Kulik, C.C. (1988). Timing of feedback and verbal learning. Review of Educational Research, 58, 79-97.

Larkin, J. H. and Chabay, R.W. (1992). Computer-Assisted Instruction and intelligent Tutoring Systems: Shared Goals and Complementary Approaches. Hillsdale, NJ: Lawrence Eribaum Associates.

Lepper, M.R., Aspinwall, L., Mumme, D. and Chabay, R.W. (1990). Self-perception and social perception processes in tutoring: Subtle social control strategies of expert tutors. In J. Olson and M. Zanna (Eds.) Self inference processes: The sixth Ontario symposium in social psychology. Hillsdale, NJ: Lawrence Eribaum Associates.

Lesgold, A., Eggan, G., Katz, S. and Rao, G. (1992). Possibilities for assessment using computer-based apprenticeship environments. In J. Regian and V. Shute (Eds.) Cognitive Approaches to Automated Instruction. Hilisdale, NJ: Lawrence Eribaum Associates.

Lesgold, A., Lajoie, S. Bunzo, M and Eggan, G. (1992). SHERLOCK: A coached practice environment for an electronics troubleshooting job. In J. Larkin and

R. Chabay (Eds.) Computer-assisted instruction and intelligent tutoring systems. Shared goals and complementary approaches. Hilisdale, NJ: Lawrence Enbaum Associates.

Lewis, M.W. and Anderson, J.R. (1985). Discrimination of operator schemata in problem solving: Learning from examples. Cognitive Psychology, 17, 26-65. Littman, D. (1991). Tutorial planning schemas. In P.

Goodyear (Ed.) Teaching Knowledge and Intelligent Tutoring. Norwood, NJ: Ablex Publishing.

Martin, J. and VanLehn, K. (1995). A Bayesian approach to cognitive assessment. In P. Nichols, S. Chipman and R. Brennan (Eds.) Cognitively Diagnostic assessment. Hilisdale, NJ: Lawrence Eribaum Associates.

Means, B. and Gott, S.P. (1988). Cognitive task analysis as a basis for tutor development: Articulating abstract knowledge representations. In J. Psotka, L. Massey and S. Mutter (Eds.) Intelligent tutoring systems: Lessons learned. Hilisdale, NJ: Lawrence Erlbaum Associates.

Merrill, D.C., Reiser, B.J., Ranney, M. and Trafton, G.J. (1992). Effective tutoring techniques: A comparison of human tutors and intelligent tutoring systems. The Journal of the Learning Sciences, 1.

Miller, M.L. (1979). A structured planning and debug ging environment for elementary programming. International Journal of Man-Machine Studies, 11, 79-95

Mislevy, R.J. (1995). Probability-based inference in cognitive diagnosis. In P. Nichols, S. Chipman and R. Brennan (Eds.) Cognitively diagnostic assessment.

Hiilsdale, NJ: Lawrence Erlbaurn Associates. Munro, A., Fehling, M.R. and Towne, D.M. (1985). Instruction intrusiveness in dynamic simulation train ing. Journal of Computer-Based Instruction, 12, 50-53.

National Council of Teachers of Mathematics (1989). Curriculum and Evaluation Standards for School Mathematics. Reston, VA: The Council.

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

Pirolli, P.L. and Greeno, J.G. (1988). The problem space of instructional design. In J. Psotka, L. Massey and S. Mutter (Eds.) Intelligent tutoring systems: Lesson learned. Hillsdale, NJ: Lawrence Erlbaum Associates.

Piroili, P.L. and Recker, M. (1994). Learning strategies and transfer in the domain of programming. Cognition and Instruction, 12, 235-275.

Poison, M.C. and Richardson, J.J. (1988). Foundations of Intelligent Tutoring Systems. Hillsdale, NJ: Lawrence Eribaum Associates.

Ramadhan, H. and du Boulay, B. (1993). Programming environments for novices. In E. Lemut, B. du Boulay and G. Dettori (Eds.) Cognitive models and intelligent environments for learning programming. New York:

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

Reiser, B.J., Copen, W.A., Ranney, M., Hamid, A. and Kimberg, D.Y. (1995). Cognitive and motivational consequences of tutoring and discovery learning. Un published manuscript.

Reiser, B.J., Kimberg, D.Y., Lovett, M.C. and Ranney, M. (1992). Knowledge representation and explanation in GIL, An intelligent tutor for programming. In J. Larkin and R. Chabay (Eds.) Computer-assisted instruction and intelligent tutoring systems: Shared goals and complementary approaches. Hilisdale, NJ: Lawrence Eribaum Associates.

Reusser, K. (1993). Tutoring systems and pedagogical theory: Representational tools for understanding, planning, and reflection in problem solving. In S. La joie and S. Derry (Eds.) Computers and Cognitive Tools. Hillsdale, NJ: Lawrence Erlbaum Associates.

Riesbeck, C.K. and Schank, R.C. (1991). From training to teaching: Techniques for case-based ITS. In H. Burns, J. Parlett and C. Redfield (Eds.) Intelligent tutoring systems: Evolutions in design. Hillsdale, NJ: Lawrence Erlbaum Associates.

Ritter, S. and Koedinger, K.R. (1995). Towards light weight tutoring agents. Proceedings of the 7th World Conference on Artificial Intelligence in Education.

Schmidt, R.A., Young, D.E., Swinnen, S. and Shapiro, D.C. (1989). Summary knowledge results for skill acquisition: Support for the guidance hypothesis. Journal of Experimental Psychology: Learning, Memory and Cognition, 15, 352-359.

Schoenfeld, A.H. (1987). What's all the fuss about metacognition? in A. Schoenfeld (Ed.) Cognitive Science and Mathematics Education. Hillsdale, NJ: Lawrence Erlbaum Associates.

Schofield, J.W. Evans-Rhodes, D. and Huber, B.R. (1990). Artificial intelligence in the classroom: The impact of a computer-based tutor on teachers and stu dents. Social Science Computer Review, 8, 24-41.

Self, J.A. (1990). Bypassing the intractable problem of student modeling. In C. Frasson and G. Gauthier (Eds.) Intelligent tutoring systems: At the Crossroads of Artificial intelligence and Education. Norwood, NJ; Ablex Publishing.

Sing1ey M.K. (1987). The effect of goal posting on operator selection. Proceedings of the Third international Conference on Artificial Intelligence and Education. Pittsburgh, PA.

Singley, M.K., Carroll, J.M. and Alpert, S.R. (1993).

Incidental reification of goals in an intelligent tutor for Smalltalk. In E. Lemut, B. du Boulay and G. Dettori (Eds.) Cognitive Models and Intelligent Environments for Learning Programming. New York: Springer-Verlag.

Shute, V.J. (1993). A comparison of learning environ ments: All that glitters.... In S. Lajoie and S. Derr)' (Eds.) Computers and Cognitive Tools. Hillsdale, NJ: Lawrence Erlbaum Associates.

Shute, V.J. and Glaser, R. (1990). A large-scale evaluation of an intelligent discovery world: Smith- town. Interactive Learning Environments, 1, 5 1-76.

Sleeman, D. and Brown, J.S. (1982). intelligent Tutoring Systems. New York: Academic Press.

Sleeman, D., Kelly, A.E., Martinak, R., Ward, R.D. and Moore, J.L. (1989). Studies of diagnosis and remediation with high school algebra students. Cogni tive Science, 13, 55 1-568.

Sleeman, D., Ward, R.D., Kelly, A.E., Martinak, R. and Moore, J.L. (1991). An overview of recent studies in PIXIE. In P. Goodyear (Ed.) Teaching knowledge and Intelligent Tutoring. Norwood, NJ: Ablex Publishing.

Stevens, A.L. and Collins, AM. (1977) The goal structure of a Socratic tutor. Proceedings of the ACM Conference. New York: Association for Computing Machinery.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. Cognitive Science, 12, 257-285.

Tenney, Y.J and Kurland, L.C. (1988). The development of troubleshooting expertise in radar mechanics.

In J. Psotka, L. Massey and S. Mutter (Eds.) Intelligent Tutoring Systems: Lessons learned. Hillsdale, NJ: Lawrence Erlbaum Associates.

Towne, D.M. and Munro, A. (1992). Supporting di verse instructional strategies in a simulation-oriented training environment. In J. Regian and V. Shute (Eds.) Cognitive Approaches to Automated Instruction. Hillsdale, NJ: Lawrence Erlbaum Associates.

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

Weber, G. (1993). Analogies in an intelligent programming environment for learning LISP. In E. Lemut, B. du Boulay and G. Dettori (Eds.) Cognitive Modelsand Intelligent Environments for Learning Programming. New York: Springer-Verlag.

Weber, G. (in press). Episodic learner modeling. Cognitive Science.

Wenger, E. (1987). Art Intelligence and Tutoring Systems: Computational and Cognitive Approaches to the Communication of Knowledge. Los Altos, CA: Morgan Kaufmann Publishers.

Westcourt, K., Beard, M and Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. Proceedings of 1977 Annual Conference, Association for Computing Machinery.

Williams, M.D., Hollan, J.D. and Stevens, A.L. (1981). An overview of STEAMER: An advanced computer-assisted instruction system for propulsion engineering. Behavior Research Methods and Instrumentation, 13, 85-90. Woolf, B.P. and McDonald, D.D. (1984). Building a computer tutor: Design issues. iEEE Computer, 17, 61-73.