DTIC FILE COPY (2)

AFHRL-TR-88-35

AIR FORCE

HUMAN RESOURCES LABORATORY

AD-A210 230

MAINTAINER'S ASSOCIATE TRAINING INSTRUCTIONAL ENVIRONMENT: ITS DEMONSTRATION (MATIE II)

Kent R. Jones
Deborah E. Bamford

Allen Corporation of America
209 Madison Street
Alexandria, Virginia 22314

J. Jeffrey Richardson
Tania M. Sizer

Center for Applied Artificial Intelligence
Graduate School of Business and Administration
University of Colorado at Boulder
Boulder, Colorado 80309-0419

Kevin J. Tiene
Kenneth R. Stephens
Thomas M. Bell
William R. Hutchison

BehavHeuristics, Incorporated
2307 Michigan Avenue
Silver Spring, Maryland 20910

TRAINING SYSTEMS DIVISION
Brooks Air Force Base, Texas 78235-5601

June 1989

Final Technical Report for Period April 1987 - March 1988

AIR FORCE SYSTEMS COMMAND
BROOKS AIR FORCE BASE, TEXAS 78235-5601

Unclassified
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY 2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFHRL-TR-88-35
6a. NAME OF PERFORMING ORGANIZATION Allen Corporation of America 6b. OFFICE SYMBOL *(If applicable)*	7a. NAME OF MONITORING ORGANIZATION Training Systems Division
6c. ADDRESS *(City, State, and ZIP Code)* 209 Madison Street Alexandria, Virginia 22314	7b. ADDRESS *(City, State, and ZIP Code)* Air Force Human Resources Laboratory Brooks Air Force Base, Texas 78235-5601
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Air Force Huma Resources Laboratory 8b. OFFICE SYMBOL *(If applicable)* HQ AFHRL	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F33615-85-C-0019/P00006
8c. ADDRESS *(City, ate, and ZIP Code)* Brooks Air For Base, Texas 78235-5601	10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
62205F	1121	10	19

11. TITLE *(Include S urity Classification)*
Maintainer's ociate Training Instructional Environment: ITS Demonstration (MATIE II)

12. PERSONAL AUT OR(S)
Jones, K.R.; Ba ford, D.E.; Richardson, J.J.; Sizer, J.; Tiene, K.J.; Stephens, K.R.; Bell, T.; Hutchison, W.R.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Final	FROM Apr 87 TO Mar 88	June 1989	50

16. SUPPLEMENTARY NOTATION

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP
05	08	
05	09	

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
artificial intelligence, Intelligent Tutoring System,
maintenance training, job aiding,
knowledge representation, computer-assisted instruction, (ST 0)

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

The purpo of this effort is the development of an Intelligent Tutoring System (ITS) to teach skills u d in electronic troubleshooting and maintenance. This system, the Maintainer's Associate Training Instruction Environment (MATIE), is based on a previously developed Intelligent Maintainer's Associate (IMA), a job-aidi expert system for maintenance of the F-111 6883 Avionics Teststand. The IMA expert system had been found be inarticulate and not adaptable for use i an ITS. Thus, a new, simulation-based expert module developed that allows the student freedom to hoose any testpoint. This simulation-based system has capability t ork with any system that can be re presented as an acyclic directed graph. Keywords:

20. DISTRIBUTION/AVAILABILITY O ABSTRACT [✓ UNCLASSIFIED/ NLIMITED] ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESP NSIBLE INDIVI UAL Nancy J. Allin, Chief, STI FO Branch	22b. TELEPHONE *(Include Area Code)* (512) 536-3877 22c. OFFICE YMBOL AFHRL SCV

DD Form 1473, JUN 86 — *Previ s editions are obsolete.*

SECURITY CLASSIFICATIO OF THIS PAGE
Unclassified

SUMMARY

The purpose of this effort is the development of an Intelligent Tutoring System (ITS) to teach skills used in electronic troubleshooting and maintenance. This system, the Maintainer's Associate Training Instructional Environment (MATIE), is based on a previously developed Intelligent Maintainer's Associate (IMA), a job-aiding expert system for maintenance of the F-111 6883 Avionics Test Station.

Two research issues were considered in the development of the ITS. First was an examination of the issues involved in adapting a previously constructed expert system to the role of an expert module in an ITS. Second was the feasibility of developing a generic ITS, *ATIE, that could deal with knowledge domains in other troubleshooting and training areas.

The IMA expert system was found to be inarticulate and not adaptable for use in an ITS. A new, simulation-based expert module was developed that allowed the student freedom to choose any test point. This simulation-based system--MATIE--has the capability to work with any system that can be represented as an acyclic directed graph.

PREFACE

This project was a collaborative effort between Allen Corporation of America and its subcontractors, the Center for Applied Artificial Intelligence (CAAI) of the University of Colorado, and BehavHeuristics, Inc. As subcontractors, CAAI and BehavHeuristics provided design and technical support.

We would like to thank the contract and technical personnel at the Air Force Human Resources Laboratory, Brooks AFB, Texas for their support. We would especially like to thank Capt Kevin Kline, Lt Col Hugh Burns, and Mr. Richard Vigue for their guidance throughout the project.

TABLE OF CONTENTS

Table of Contents (Concluded)

LIST OF FIGURES

MAINTAINER'S ASSOCIATE TRAINING INSTRUCTIONAL ENVIRONMENT: ITS DEMONSTRATION (MATIE II)

I. INTRODUCTION

Current Situation in Maintenance Training of Military Systems

Maintenance is an important logistical consideration in any deployed weapon system, comprising as much as 50% of life cycle cost (Lahore, 1984). Diagnosis is a major aspect of maintenance and includes the tasks of fault detection and fault isolation. As systems become more complex, the natural tendency is for maintenance costs (in dollars as well as in lost availability) to increase. Many current initiatives are working to decrease maintenance activity and improve weapon system availability. Improved reliability, testability, and maintainability are crucial elements of improved availability.

Recently, the Department of Defense and all military services have adopted a systems engineering approach to diagnostics termed *integrated diagnostics* (Nationai Security Industrial Association, 1986). The integrated diagnostics philosophy recognizes that issues relevant to optimal diagnostic capability exist in all phases of system development--conceptual, demonstration and validation, full-scale development, and deployment. Further, it recognizes that there are several major diagnostic resources that must be used in the proper mix, depending on the situation--design for testability, built-in and self test, off-line automatic test, technical documentation, and traineo personnel.

History has shown that over-reliance on one diagnostic resource is usually suboptimal. Clearly, today's systems could not be maintained by trained personnel alone without assistance from automated test resources--the systems are just too complex. But the extreme response of over-reliance on automatic test equipment at the expense of trained personnel has led to problems when the automatic test equipment did not have the test coverage expected and sufficient numbers of trained personnel were not available to compensate for the difference. Over-reliance on built-in test created maintenance problems when built-in test systems had inordinately large false alarm rates. An appropriate mix of diagnostic resources must be identified for each new system.

Job Requirements for Maintenance Personnel

As a diagnostic resource, maintenance personnel will probably always be an element of the diagnostic mix in logistics support. At a minimum, trained personnel are the diagnostic resource of last resort. When reliability fails, when design for testability fails, when built-in test fails, and when automatic test fails to isolate a fault, trained personnel must fill the breach.

The job requirement of maintenance personnel is changing. The forces behind these changes include the increasing technological complexity of systems and the decreasing demographic availability of recruits. Training technicians to have the knowledge and skills for the total array of technologically complex systems they may encounter is not feasible, if not impossible. Thus, technicians need to be able to augment general understanding through use of technical documentation when automated diagnostic systems fail. This requires generalists, not specialists--the thrust of the RIVET WORK FORCE initiative in the Air Force. Demographic scarcity also calls for generalists, since there is a growing variety of specific diagnostic tasks, yet fewer people available to work on them. In diagnostics, the most important skill a generalist can possess is an understanding of how to approach troubleshooting tasks.

To summarize, due to increasing technological complexity, demographic scarcity of recruits, efforts to maximize system availability, and a systems engineering approach to diagnosis, trained maintenance personnel with generalist skills will continue to be an important diagnostic resource in the Air Force. Thus, maintenance training is and will remain an important aspect of logistics support.

Intelligent Tutoring Systems

Foundation of ITS

The potential advantages of the computer for automated and individualized instruction have long been recognized, but only in recent years has computer-based training been enhanced by the incorporation of innovations from the field of artificial intelligence (AI). Research into ways that AI can add to the flexibility and effectiveness of the computer as a teaching aid has produced a new and exciting field of automated instruction that has been called Intelligent Computer-Assisted Instruction (ICAI) and, more recently, Intelligent Tutoring Systems (ITSs). As the number of completed ITS projects increases and the lessons learned accumulate, the ITS appears to have moved past its infancy and represents a viable technology that will soon complement, supplement, and compete with other forms of training.

The earliest project that explicitly attempted to incorporate AI into instructional software was the SCHOLAR project reported by Carbonell (1970). SCHOLAR was really the beginning of a new paradigm, and set off in a much different direction than most of the Computer-Assisted Instruction (CAI) or Computer-Based Training (CBT) tradition that existed up to that point. There was, and continues to be, some blurring of the otherwise crisp distinctions between the ITS and more traditional CAI. Some of the work completed in the early 1970s by Atkinson (1972, 1976) and Suppes, Fletcher, and Zanotti (1976) that was called adaptive or generative CAI had some elements in common with the ITS, but modeling of student progress in these studies relied primarily on statistical techniques rather than methods normally associated with AI. It was not until the late 1970s that many of the pioneering ITS projects began to be reported in the literature. The book by Sleeman and Brown (1982) was a major milestone in the field, and discussed less than a dozen examples of ITSs. Several new books together mark the maturation of the field. A recent book by Wenger (1987) discusses roughly three times as many examples of ITSs as were commonly cited in 1982. Other books by Polson and Richardson (1988), Kearsley (1987), and Psotka, Massey, and Mutter (1988) discuss the most recent advances in the field.

ITS and Traditional CBT

The ITS shares some of the objectives of more traditional CAI or CBT, but comes from a different tradition and hence approaches its task differently. Where CBT traces its heritage to Instructional Technology and Programmed Learning, the ITS has its origins in Computer Science and Cognitive Psychology, which have come together in the field of Cognitive Science. While the origin of the ITS outside of the older CBT tradition has given ITS researchers the flexibility of approach that comes with intellectual autonomy, it has also made it difficult to incorporate many of the lessons learned from over two decades of CBT, and many years of programmed instruction prior to that. Park, Perez, and Seidel (1987) described the strengths of both movements and also characterized an unfortunate failure to synthesize desirable features of the two movements which results from adherence to one paradigm or the other.

Park et al. (1987) characterized the separation of paradigms along several lines:

1. With its longer history of lessons learned, CBT's goals are now more practical and outcome-oriented. The goal of many ITS studies has been to explore the technical aspects of building intelligence into the instructional system rather than focusing on instructional or domain features.

2. Largely due to limitations in the flexibility of authoring and delivery software tools and its teaching machine heritage, CBT has relied on a mode of instruction in which the student is given no initiative. The CBT system administers a relatively inflexible sequence of "frames," which every student experiences in the same way. "Branching" (Crowder, 1959) is a technique which varies the sequence on the basis of student performance, but still the CBT system determines what the student sees next in each case. The ITS has provided a much more flexible "mixed-initiative" mode of interaction, in which the system exerts some control over what is to be learned, but the student can take control as well to pursue individual learning objectives and local interests. The ITS can be characterized as a "student-centered discovery" method.

3. CBT has a well-defined methodology for developing instructional material, which has resulted in a steady increase in the quality of CBT. The Instructional Systems Development (ISD) model is a formalization of these methodologies drawn from instructional technology (AFM 50-2, 1975). It is basically the result of the application of the systems approach to training. CBT's front-end analysis often includes a task and/or concept analysis, definition of performance algorithms, and identification of hierarchical relationships in the subject-matter domain. Several well-defined steps are involved in the life cycle of CBT software: analysis, design, development, formative evaluation, implementation, summative evaluation, and maintenance. No such standard methodology holds for the ITS. The knowledge structures that are employed in the ITS vary with whatever is implied by the particular knowledge representation scheme used by the implementors. Neither are project management and product development plans rigorously defined in the ITS. As in other AI work, where large complex software systems are developed by an individual or a small team, the development methodology of choice has been "exploratory programming" and incremental refinement. There has been little emphasis on evaluation of outcomes.

4. There are several types of CBT teaching modes: drill- and-practice, tutorial, games, and simulations. The ITS makes heaviest use of tutorials and games, but these are dissimilar to their CBT counterparts.

5. CBT's tutorial style is mainly expository, with an attempt at teaching by presenting rules and definitions, followed by a carefully chosen set of examples and nonexamples of concepts. Good CBT engages the student in active responding, but the degree of this interaction depends on the imagination and technical capabilities of the designers. At its worst extreme, CBT is an "automated page-turner"; at its best, CBT involves the student in relevant active responding with good performance feedback. The ITS has a different style of tutorial interaction based on question-and-answers partly initiated by the student. Often, the pedagogical model in the ITS is that of the Socratic method.

6. Games in CBT are used for teaching gaming skills and rules or to provide external motivation for some other learning. ITS games are intended to be a "reactive learning environment" (Brown, Burton, & Bell, 1975) employing a style of tutorial interaction called "coaching."

7. CBT tends to make its judgments of the student's performance on the basis of discrete binary responses. Where more sophisticated student modeling has been attempted, it has used mathematical models, statistics, and regression analysis. The ITS has modeled the student qualitatively, relying on the developer's subjective judgments. The student modeling modules in ITS are discussed in more detail below.

We will return to some of these issues in a later discussion of future directions for ITSs.

Characteristics of Intelligent Tutoring Systems

Kearsley (1987) identified five different types of intelligent tutoring systems:

1. coaches - the student's performance in a simulation or game is the stimulus for ITS interruption for the purpose of giving controlled advice and asking leading questions;

2. mixed-initiative tutors - the ITS software engages the student in a two-way conversation using the Socratic method;

3. diagnostic tutors - the ITS observes the student's process of solving problems and identifies certain kinds of "bugs" (either "malrules," misconceptions, or missing conceptions);

4. microworlds - graphic simulations allow a student to explore a problem domain;

5. articulate expert systems - expert systems used as job performance aids can have the ability to explain their actions in a way that can be used for training.

ITSs are generally characterized by the presence of a modular approach which includes a sophisticated student modeling module, a functioning expert capable of solving the problems in the subject domain, and a module that is capable of managing the dialog between the student and the system as an expert tutor might do. In some cases, the dialog between student and ITS is carried on in conversational English through natural language techniques. The expert module is generally "articulate" -- it has the capability of generating explanations for what it is doing and why. The instructional module has capabilities for matching the presentation of new material and the degree of tutorial intervention to the level of progress of the student. These features and others are discussed in more detail below.

Architecture of ITS

The classic structure of an ITS separates functions into three independent modules that work together. These have most commonly been called the domain expert model, the student model, and the instructional model. Other modules for curriculum management and instructor interaction complement these.

The Domain Expert Model. Whereas domain knowledge within traditional CAI is embedded within the tutorial structure of the courseware, the domain knowledge of an ITS is separate from the tutorial process. The expert model contains the expertise to be learned by the student, and often the ability to solve problems in the subject domain. Not only is it able to solve problems as an expert would, it is also able to solve the problem in multiple ways to give it the flexibility to handle multiple approaches to the problem that students might attempt. The domain expert is closely related to the kind of expert system which would serve as a job performance aid in a non-instructional application. However, the demands on this expert model are greater than on typical expert systems; so, it has been found in several ITS applications that an existing expert system is not necessarily suited without modification to be the ITS domain expert (Clancey, 1987). Knowledge that may have been implicit in the structure of the expert system has to be explicated to serve the instructional purpose. A knowledge structure that is adequate for a job performance aid may be unsuited to providing explanations or serving other pedagogical purposes.

Knowledge is organized in the domain expert by various means in different ITS projects, depending on the nature of the problem domain. Among the knowledge representation schemes employed have been semantic nets, trees, lattices, procedures, predicate logic, and production rules. The knowledge structure encoded in the expert model serves as a baseline for comparison with the student modeling module in some systems.

The expert model can be either opaque (able to solve the problem, but unable to discuss its methods directly) or articulate (with natural language capabilities to explain and discuss the knowledge base). For many uses, it is preferable to have an articulate expert, but of course this capability adds to the development effort.

Anderson (1988) refers to opaque experts as black boxes and to articulate experts as glass boxes. A step beyond expert systems according to Anderson's taxonomy are the "cognitive systems" which incorporate human-like knowledge of three types: procedural, declarative, and qualitative. Procedural knowledge is that which is required to complete a task; for instance, how to perform integration in the domain of calculus. Declarative knowledge is represented by a set of facts to be learned in a domain such as history or geography. Qualitative knowledge about causal relationships has been involved in many of the ITSs to be discussed below.

The Student Model. The student model keeps track of topics, concepts, and skills the student has mastered during tutoring. It may also keep track of other student information, such as the preferred mode for interacting with the tutor ("cognitive style"), the general level of ability, what the student seems to forget easily, and the student's own goals and plans.

There are several types of student models. An overlay model is one in which the student module is simply considered to be a subset of the expert model. A differential model focuses more on the critical differences between the student and the expert. There are two types of "buggy" models in which the focus is on the errors the student makes. In the first, the perturbation model, the student modeling module is considered to be an incorrect variant of the expert model. In the second, termed the "mal-rules model," incorrect rules have been substituted in the expert model.

Several sources of evidence are used to maintain and update the student modeling module: implicit evidence from problem-solving behavior, explicit evidence from direct questions asked of the student, and historical evidence either within a tutorial session or across multiple sessions.

There are some problems with inferring this evidence. There is the classic problem of credit attribution; that is, designating some element of the student modeling module as responsible for a particular error made by the student. A second problem is that the system may assume too much or too little student knowledge. The usual solution for this is to carefully construct the exercises and examples to help diagnose the source of error. A third problem is noisy data. Student mistakes come not only from systematic misunderstanding but also from other incidental factors such as fatigue, boredom, and cognitive overload. Finally, as Sleeman and Brown (1982) pointed out, attempts to infer the student's plan (procedures or problem-solving strategies) are combinatorially explosive.

VanLehn (1988) has identified nine different techniques that have been used to diagnose student performance. T..ese may be paraphrased as follows:

1. model tracing - an "underdetermined interpreter" suggests a set of rules that might be applied next during problem solving; the student's actual response determines which rule was applied and thereby suggests the nature of any error that occurs;

2. path finding - often used with model tracing, the chain of rules applied to cause transition between two states (the "path") is used to infer the rules; the problem is that this path is not always unique;

3. condition induction - used with "buggy" student modules (which concentrate on the types of errors the student makes rather than looking at the student repertoire as an incomplete portion of the expert's knowledge), a bug part library is used to induce the rule that connects one state with the next;

4. plan recognition - when knowledge is procedural and hierarchical, problem solving is adequately described as a tree or "plan." Plan recognition is the process of recognizing the plan from a few "leaves" on the tree, so that model tracing can proceed;

5. issue tracing - a variant of model tracing, issue tracing considers concepts or components of skills to be issues; issue tracing monitors issues that have been correctly used or misused;

6. expert systems - this approach uses diagnostic rules to evaluate whether rule-based knowledge has been learned;

7. decision trees - used with bug libraries, a tree is constructed in advance so that student errors can quickly be diagnosed as to the conceptual bugs that account for them;

8. generate and test - given a set of candidate bugs, bug pairs are formed to select for those multiple bugs that best account for student errors;

9. interactive diagnosis - the system chooses problems that give the best diagnostic information for suspected bugs.

Ideally, the student modeling modules should also include a "critic" function to measure how predictive of student behavior the model really is, and to suggest corrections along the way.

The Instructional Model. The instructional model manages the process of tutorial interaction with the student. Dialog management heuristics are employed to control these interactions. The instructional model sometimes has natural language capabilities that allow it to engage the student in dialogs. It represents the teaching methods and strategies, and has the responsibility to decide when to interrupt the student, what to say, and how to say it. Often this means making constructive use of student errors. ITSs designed for teaching declarative knowledge have often adopted a Socratic model of tutoring.

In coaching systems, the instructional model acts as a coach who engages the student in an activity such as a game, as in WEST (Burton & Brown, 1982). The coach observes the student's play and occasionally interrupts to give new information or suggest new strategies. Closely related is the notion of a reactive learning environment, in which the student "learns by doing" in a simulated environment. This approach allows experimentation outside of the actual environment in which task performance will later occur. The advantages of conducting this experimentation in a simulated environment are that it makes experimentation easier and safer for the student, and allows students to learn from their mistakes without the disastrous results that might result from such mistakes in the actual work environment (Brown, Burton, & de Kleer, 1982).

Instructor Interfaces. The same supportive software environment that allows developers to interact productively with training material as it is being developed also has been applied to support instructors in developing new exercises. For instance, STEAMER and IMTS (discussed in more detail below) both support graphic interfaces that can be used to combine standard elements into new configurations that can be used for problem-solving and troubleshooting. OBIE, an Object-Based Intelligent Editor (Freedman & Rosenking, 1986) makes use of material prepared in an authoring process that involves a Graphics Interface Editor, a Cognitive Connection Editor, and a Panel Sequencing Editor. There are several other examples in the literature of utility functions developed around the object orientation of an ITS to support construction of new exercises from a library of standard components.

A second type of interface for the instructor has as its purpose the summarizing of student performance, either of an individual in one or more training sessions, or of a group of students. This is an important function that corresponds with Computer-Managed Instruction in more conventional CBT.

Curriculum Management

The selection and sequencing of material to be covered by the tutorial is an area where ITS projects should be able to make more use of prior research in educational technology, and specifically, methodologies like the Instructional Systems Development (ISD) model. Halff (1988) has presented a treatment of the problems of curriculum development and management.

Expository tutors using natural language techniques to conduct a Socratic dialog with the student have made use of the approach of "web teaching" in which a knowledge structure much like an Augmented Transition Network (ATN) organizes concepts, and the tutor chooses the next topic on the basis of relatedness and generality. One particular ITS, Meno-tutor, has combined an ATN approach called the Discourse Management Network (DMN) with meta-rules that modify the network's path of instruction to reflect the current situation with the student.

Halff (1988) listed three major principles for tutoring based on examples and exercises. The first principle, *manageability*, is that exercises should be sequenced such that every student who has completed the material leading up to an exercise should be able to complete the new material. There has been a good deal of emphasis in instructional technology on this principle, including Gagne's (1977) work on enabling objectives, and Merrill's (1977) work that suggests choosing examples to demonstrate both the range of a concept (divergence) and the critical differences between examples and non-examples that determine classification (matching).

The second principle, *structural transparency*, states that the sequence of examples and exercises should reflect the structure of the procedure being taught and thereby help communicate it to the student. The "step theory" of VanLehn that suggests sequencing material into discrete steps of manageable difficulty is one approach to this principle; in fact, it is a principle that was presented very early in the history of programmed instruction (Skinner, 1968). In educational technology, Englemann and Carnine's (1982) method of "direct instruction" suggests well-defined methods of sequencing material based on features of the concepts to be taught.

The third of Halff's principles, *individualization*, suggests taking into account the student's current pattern of skills and weaknesses when determining the instructional sequence. This principle was also recognized some time ago in the field of instructional technology, but the tools to properly apply it were lacking until the recent work with ITSs.

Halff and others have pointed out that work in ITS has not made contact very well with other work in the fields of education and instructional technology. This is one of the undesirable effects of the ITS's developing as a separate paradigm. Some suggestions for making better use of the lessons learned in the existing instructional literature will be discussed later as future directions for research in ITSs.

ITS and Maintenance

Because of the strengths of ITS for establishing a reactive learning environment with simulations and models, maintenance has been one of the most frequent topics for ITSs. Working models of a complex system can be presented graphically, so that students can see the effects of manipulating elements of the system and can make simulated measurements at various points in the system to troubleshoot problems. With a few exceptions, the ITSs within the maintenance domain have been for electronic circuits.

Cost-effective methods of providing maintenance training have been actively sought in Department of Defense research programs for decades. The replacement of actual equipment with simulated equipment has been the basic approach taken. Simulators were shown to be a less expensive and equally effective way of training than use of actual equipment (Orlansky & String, 1983). As the simulations became increasingly sophisticated and emphasized cognitive fidelity over physical fidelity, cost savings were coupled with training effectiveness gains (Pieper, Richardson, Harmon, Keller, & Massey, 1984). The cognitive fidelity of a training system is the degree to which the system replicates the actual mental steps involved in competently solving a problem.

Intelligent tutoring system technology provides the most sophisticated approach to maintenance training. An ITS is a dynamic learning environment with high cognitive fidelity, that actively applies instructional principles to maximize trainee learning (see Polson & Richardson, 1988). Thus, an ITS has an "expert module," containing expertise in the problem-solving domain; a "student modeling module" that tracks differences between expert performance and student performance; and an "instructional module" that contains the instructional principles used to efficiently reduce the expert/student differences. Figure 1 illustrates these basic relationships. Coupled with these modules is an instructional environment which provides the context of learning. In the case of troubleshooting training, the instructional environment is most often a diagram or schematic of the system under test, with which the trainee interacts in a fashion similar to a real troubleshooting situation by making measurements and hypothesizing faulted components. ITS technology will be discussed in further detail below.

Approaches to the expert module of maintenance ITSs are also found in the literature on diagnostic expert systems. Almost every diagnostic expert system investigator points out the instructional utility of the diagnostic job aid. Conversely, builders of intelligent maintenance training systems point out the related potential applications of providing guidance in automated diagnosis and design for testability. More often than not, the same people are working both sides of the problem. The reciprocal relationship between job aids and trainers can be seen as follows: A job aid states where to measure and asks for a measurement result; a trainer asks where to measure and states the measurement result (Figure 1).

To summarize what has emerged over the past 5 years in this field, a consensus view has formed that the diagnostic expert system must be based on a *model* of the equipment under test. This model includes the device's components, the components' attributes (behavior, reliability, cost and testability) and interconnections, all organized hierarchically. Components can be part of any hardware technology--electronic (digital, either combinational or sequential), electrical, mechanical, electro-mechanical, hydraulic, or pneumatic. As long as the components (and their attributes) and their interconnections can be identified, the model-based method applies.

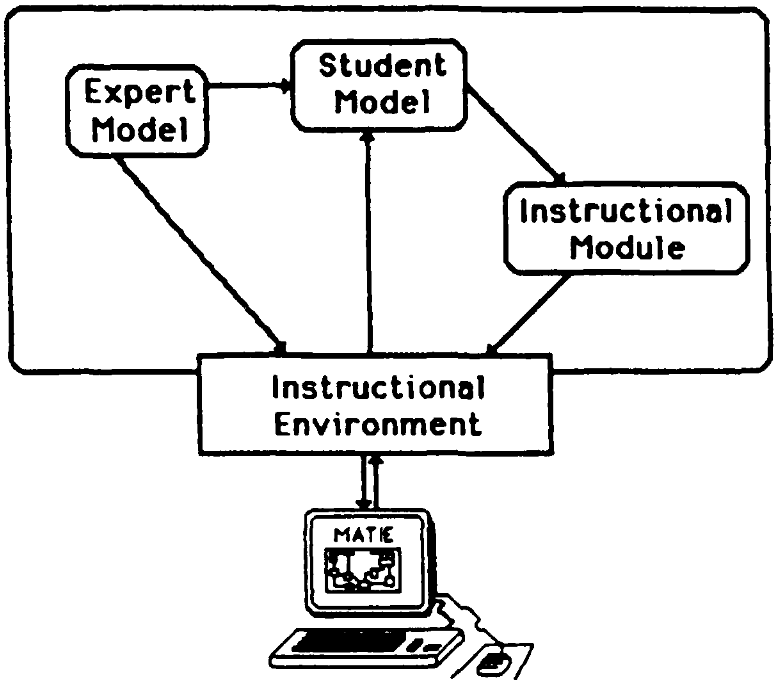


Figure 1. **Components of an Intelligent Tutoring System (ITS).**

There are numerous ways of implementing a model-based troubleshooting system (see Richardson, 1987, for an extensive bibliography). Fundamentally, the model is used to generate expected measurements. Differences between expected measurements and observed measurements are used to propagate blame or innocence along the paths of connectivity in the device. Then a new measurement is selected by computing the expected information gain per unit cost of all possible test points and selecting the one with the maximum gain.

The model-based approach is in contrast to early expert systems work based on chaining through symptom-fault rules collected from subject-matter experts by knowledge engineers. In contrast to this evidential, rule-based approach characterized as first-generation expert systems, the second-generation, model-based approach has several advantages.

The main virtue of the model-based approach is device-independence. Diagnostic expertise is not based on symptom-fault rules garnered from experience and listed in advance, but rather, on a device-independent algorithm capable of generating tests and deducing their implications based only on knowledge about the components and their interconnections. Device-independence permits efficient construction of model-based systems, since only the data involved in the device's topology and behavior need be supplied. Detailed empirical rules are not needed.

A second virtue is robustness. The first-generation systems are subject to a phenomenon termed "the knowledge cliff." That is, their performance drops precipitously when the problems presented lie close to or outside of the realm of expertise of the system. Furthermore, performance within the circumscribed domain may be uneven, due to missing or erroneous rules. The model-based approach is more robust because its performance is based directly on a description of the device under test. To a large degree, if the description is complete, performance will also be complete. Validation and verification, the issue of the completeness and consistency of the rule base, are not a problem because there is no rule base. Validation and verification become a matter of confirming the completeness of the device model. The device model is much easier to confirm than a rule base because the model is data--the specification of the unit under test--and is known in advance.

How valid is this device-independent algorithm as the basis for cognitive fidelity in maintenance ITSs? Studies of human troubleshooting behavior have shown that people employ both model-based and symptom-based methods (Keller, 1985; Rouse, 1985). The model-based method is used to solve novel, unfamiliar problems. But once troubleshooting problems associated with a specific piece of equipment or class of equipment are confronted repeatedly, people learn to recognize characteristic symptom-fault associations and begin to solve problems through the symptom-based approach akin to pattern recognition. Thus, the model-based approach is a reliable approach, but it takes effort and time to apply. Whenever possible, experts avoid it. However, when the preferred symptom- based approach is of no avail, as in novel, previously unencountered situations, the expert reverts to the methodical model-based approach.

From this discussion, it is evident that human performance in diagnostic reasoning involves, depending on circumstances, a combination of symptom-based and model-based reasoning; neither is relied on exclusively, and both play important roles. AI researchers have implemented hybrid diagnostics approaches, combining first- and second-generation expert system methodologies in one integrated system. IDM (Fink, Lusth, & Duran, 1984) and AI-FERRET (developed at Hughes Aircraft and Missile Company) have both implemented hybrid approaches. Figure 2 illustrates the features of model-based and symptom-based troubleshooting techniques, as well as the features of the hybrid approach.

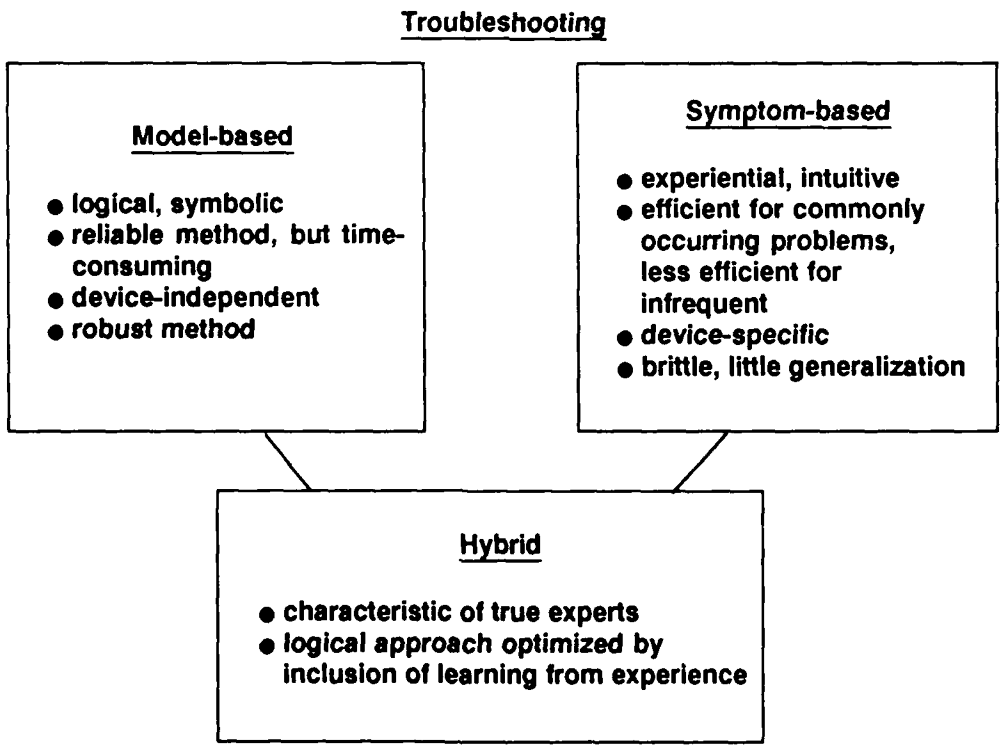

Troubleshooting
Model-based
• logical, symbolic
• reliable method, but time-consuming
• device-independent
• robust method
Symptom-based
• experiential, intuitive
• efficient for commonly occurring problems, less efficient for infrequent
• device-specific
• brittle, little generalization
Hybrid
• characteristic of true experts
• logical approach optimized by inclusion of learning from experience

Figure 2. **Model-based and symptom-based troubleshooting approaches are applied by experts in different circumstances.**

For the purposes of training, the model-based approach is preferred as the initial focus because experts rely on this approach as the basis for building up symptom-based troubleshooting expertise in the form of symptom-fault associations. They also revert to this approach when confronted with novel situations. Thus, learning to troubleshoot "from first principles" is a prerequisite to expert-level performance and necessary in support of the robustness characteristic of expert performance in novel situations. More advanced training would involve the trainee in the hybrid diagnostic methodologies, but should do so only after initial mastery of the basic model- based approach.

SOPHIE. SOPHIE (the SOPHisticated Instructional Environment) is a reactive learning environment for electronics troubleshooting, developed with funding from the Air Force Human Resources Laboratory (AFHRL). SOPHIE has undergone several revisions since it was begun by John Seely Brown and Richard Burton in 1973. At that time, Brown and Burton were at the University of California at Irvine, but shortly thereafter, their work moved with them to Bolt, Beranek and Newman (BBN) in Cambridge, Massachusetts. Later, they and the project moved to Xerox PARC (Palo Alto Research Center). The first version, SOPHIE I, made use of an enhanced model of the SPICE electronic simulator, a semantic network of information about the IP-28 power supply, and a natural language dialog mechanism (Brown, Burton, & Bell, 1975). SOPHIE I was noteworthy because its use of multiple knowledge representations (simulation-based, procedural, and declarative knowledge) provided a robust inference engine.

SOPHIE II was an enhancement which contained an expert troubleshooting module. The student could insert a fault at any point in the circuit and watch the system as it located the fault. The expert module was "articulate" and could explain the strategies it employed along the way. However, because it did this by following a fixed decision tree in which pre-stored text provided explanations of the qualitative reasoning behind its troubleshooting, it was not able to follow and coach the student.

The approach of SOPHIE III was somewhat different, employing a game format. This necessitated the development of a powerful coaching function with capabilities of expert reasoning in the domains of games and elementary mathematics. To facilitate the actions of the coach, SOPHIE III featured a student modeling module. SOPHIE III also had an expert electronics reasoner. This component is noteworthy because it recognized a necessity to include both generic knowledge of electronics principles with more device-specific knowledge (with emphasis on the generic). The system's troubleshooting expert operates in conjunction with an electronics expert: The former evaluates hypotheses and proposes measurements; the latter analyzes the circuit using a combination of basic electronics principles, component models, and device-specific behavior trees.

Several kinds of knowledge were identified. Troubleshooting strategies are heuristics that influence the sequence of testing. The "half-split" strategy of roughly dividing the search space in a circuit with each test is an example of such a strategy. On the other hand, troubleshooting tactics involve other factors (e.g., the effort required to make a particular test or the prior probabilities of failures in a particular component) which combine with and complicate the global heuristics. Brown, Burton, and de Kleer (1982) pointed out that good troubleshooters also have knowledge of circuit laws such as Ohm's Law or Kirchoff's Law, knowledge about characteristics of circuit components, and an understanding of the function of the device on which they are working. Two other types of troubleshooting knowledge were also identified: "passive" troubleshooting, in which the information about the correctness of measurements is assimilated; and "active" troubleshooting, in which decisions are made about the next measurement to make.

STEAMER. The STEAMER project, which began in the late 1970s at the Navy Personnel Research and Development Center (NPRDC), was conducted with the assistance of researchers from BBN. The task domain is the maintenance of a large complicated steam propulsion system that Navy engineers might encounter onboard large ships. STEAMER makes use of a quantitative model which determines the actions of an interactive inspectable simulation of the system. Since the components and piping of a steam system are distributed over such a large area of the ship and visually obscured by bulkheads and other equipment, it is difficult to learn to deal effectively with the system as a whole instead of small subsystems out of context. STEAMER's graphics views are thus a valuable aid to understanding complex system dynamics. The underlying quantitative model contrasts with the interest in qualitative models in such systems as SOPHIE and QUEST (see below).

Bit-mapped graphic objects called icons represent components of the steam system. The student can use a mouse (hand-held pointing device) to manipulate and select these icons in simulating maintenance and testing of the system. The intelligent graphics interface is made possible by an object-oriented extension to the LISP language called Flavors. A graphics editor is available for use by the instructors to manipulate icons and tailor the system to their own needs.

Unlike some ITS projects that allow free student exploration through a knowledge domain, STEAMER has as one of its learning modes the training of standard operating procedures. This curriculum has more structure than some other ITS approaches. Another mode is the feedback minilab in which the students can construct abstract systems out of the graphics icons and watch the dynamics of these systems as a consequence of manipulations of subsystem components.

QUEST. Qualitative Understanding of Electrical System Troubleshooting (QUEST) is a continuation of the interest at BBN with qualitative mental models for guiding ITS tutorials. Barbara White and John Frederiksen are the principal investigators on this project. The qualitative reasoning and mental models are heavily influenced by the work of Brown, Burton, and de Kleer (1982) for the SOPHIE project.

In tests with seven high school students who had no real understanding of even basic concepts of voltage and resistance, a progression of mental models of increasing sophistication and difficulty was used to gradually shape the students' understanding of circuit dynamics (White & Frederiksen, 1986a, 1986b). "Zero-order" models in which the student had only to identify whether a light would come on depending on switch settings and circuit paths were employed initially. Later, a first-order model was introduced requiring the student to answer questions about whether voltage would increase or decrease, given certain changes in the circuit. Models differed not only in terms of zero-order versus first-order, but also in the degree of elaboration. Each mental model was associated with its own version of the student modeling module, the domain model, and the instructional module.

Before students could benefit from the explanations of this articulate system, basic concepts of voltage, resistance, current, device state, internal conductivity, and series versus parallel circuits were defined. Explanations could have been given either in terms of current flow or voltage drop. The authors asserted that giving the students both would be confusing, so they chose to have the system consistently explain things in terms of voltage drop.

The QUEST system architecture allows several modes of instruction. The qualitative mental models for each circuit are runnable and inspectable. The students can create new problems using a circuit editor. The system can turn a problem into an example by solving it for the student. Initiative in QUEST can be with either the student or the system, so that several learning strategies can be employed by students: open-ended exploration, problem-driven learning, example-driven learning, and student-directed learning. In the study with high school students, subjects let the system take the initiative most of the time and seldom used an exploratory approach. This may be an artifact of their own recent experience with typical public school teaching methods, or it may indicate that the students "didn't know what to ask"--always a risk in student-directed teaching.

Several issues surrounding the use of multiple models have surfaced in the QUEST work. One limitation of the use of qualitative models is in simulating changes through time; sequences are represented in ordinal, not interval, time. Several results of model transformations were identified: knowledge acquisition, refinement, generalization, differentiation, and integration. The set of possible model transformations is very large and varies from learner to learner. As a result, QUEST was implemented with only a linear progression of models. The experience in QUEST so far has demonstrated the importance of development of these multiple models with maintainability and modifiability in mind. Knowledge structures in the models may be changed in the following ways: addition, deleting, rewriting, refining, and revising control knowledge.

The troubleshooting algorithm in QUEST used the split-half technique only once, and then shifted to a different method of isolating the fault within half the circuit: a serial search method to examine each remaining component sequentially. Students were able to out-perform this algorithm given some experience. In a critique of the effectiveness of the system, the authors concluded that there should have been more problems that asked the students to identify all of the possible faults consistent with a particular test behavior.

OHM. An object-oriented ITS shell developed at the Learning Research and Development Center (LRDC) of the University of Pittsburgh is called the Bite-Sized Tutor (Bonar, Cunningham, & Schultz, 1986). One of several ITS applications of this shell is OHM, for training basic electrical concepts.

The rationale for the Bite-Sized Tutor is that most ITS implementations are complex and unwieldy and can benefit greatly from the organization and inheritance mechanisms of an object-oriented software approach. Current systems are not very modular, and component code about the various ITS functions is distributed throughout the system and is repeated in several places. When knowledge is organized in an obscure manner, it is difficult to re-use and extend parts of one tutor for a separate application. Unfortunately, the first-generation applications of the Bite-Sized Tutor do not share very much generic code.

In building a tutor based on LRDC's approach, the first step is to analyze the domain and observe novices while they are learning. A "bug catalog" and a "cognitive task analysis" result from this initial step. Topics and issues are organized in the system as instances of a class of objects in the system called Bites. The inheritance mechanisms of LOOPS (the Xerox object-oriented extension to LISP) ensures that common information is made available to all Bites, and that the appropriate subclass of the Class Diagnoser and other system components is available to each Bite.

There are several subclasses of Class Bite. Abstraction Hierarchy Bites represent an ordering of the concepts within a curriculum. Each of these that is relevant to a particular student response is checked by the Diagnoser to see if the student is using concepts correctly. Definition Bites represent concepts the student is to learn without much background. Input/Output (I/O) Bites are concepts that primarily exhibit black-box behavior. Discovery Bites detect when the student is floundering in the exploration of simulation microworlds, and respond appropriately.

The Student Model makes use of information provided by the Diagnoser and the class variables of the Class Bite to maintain records of the events of the session. Student performance is modeled three ways: across an entire session, over only the most recent five events, and on the basis of the last event.

Several alternative types of Tutoring Mode track the local state of instructional tasks, and monitor criteria for satisfaction of the mode or detection of student floundering. The types of Tutoring Mode are Exploration, Experimentation, Elaboration, Didactic, Demonstration, and Coaching.

IMTS. Douglas Towne and colleagues at the Behavioral Technology Laboratories at the University of Southern California have developed a generic model of troubleshooting, called PROFILE, which has also been combined with an ITS model (Towne, 1986). The troubleshooting model bases its decisions of test points and evaluations of a student-chosen course of action on an estimation of the corrective maintenance time, derived from system design and maintenance condition knowledge. PROFILE has a library of system components with associated information about reliability, cost, and time to replace. It also has a second library containing times to make specific test readings and to set up test equipment. The value of each test is the new information generated by that test divided by the performance time. New information is calculated by the classic "entropy" formula in information theory. A "good test" using this information-based approach is one that partitions the most highly suspected components into different sets with the least effort. Half-splitting is a special case of this approach.

A Computer-Aided Design/Computer-Aided Engineering (CAD/CAE) system is used in conjunction with PROFILE to provide a graphic representation of fault effects and system function during training. The training system is called Intelligent Maintenance Training System (IMTS). The graphic orientation of IMTS reflects the influence of the STEAMER project, since IMTS was also funded by NPRDC. Like STEAMER, IMTS has a graphics editor that can be used by the domain expert to build simulations from a library of generic objects or icons. IMTS uses the mechanisms of PROFILE to evaluate a student's troubleshooting skills, and interrupts when problems are detected or when specific advice is called for. An interesting aspect of the IMTS student modeling module is that each component that can fail is assigned a subjective level of difficulty, and the student modeling module keeps track of the malfunctions the student has solved and their difficulty.

The first application of IMTS is as a trainer for fault isolation in the Bladefolding system of an SH-3H helicopter. In this application, IMTS is supplemented by an interactive videodisc simulator called Generalized Maintenance Training System (GMTS).

HAWK MACH III. A current project of BBN for the Army Research Institute (ARI) and the U.S. Army Air Defense Artillery School (USAADASCH) is MACH III (Maintenance Aid Computer for Hawk Intelligent Institutional Instructor), an ITS for training maintenance of a High-Powered Illuminating Radar (HIPIR) component of the HAWK missile system. The training is targeted toward the HAWK Firing Section Mechanics. HIPIR is a heterogeneous system composed of electrical, mechanical, and electronic components within a unit the size of a small truck. MACH III is an articulate expert that simulates the subsystems of HIPIR. It supplements a 39-week program of instruction, in which hands-on time is very limited. There are eight trainees per instructor during these times. MACH III helps by making a simulation of HIPIR available to individual trainees.

Maintenance is assisted by the presence of built-in test equipment (BITE). The BITE subsystem injects standard test signals into certain critical sections of circuits and measures the outputs. When the outputs are out of an acceptable range, this is reported to the operator. However, the BITE does not report the specific measurement but instead, reports one of 50 distinct categories of fault indications labeled by their corrective actions. These actions correct less than half of the faults. Interpreting BITE indications is very difficult for the operators, partly because the way the tests work is something of a mystery to the operators and because BITE itself is sometimes unreliable. The primary references for the HIPIR are also very difficult to use, and are not always correct.

The troubleshooting doctrine taught to these mechanics is noteworthy. A fault usually is first called to their attention by a phone call from the field. If it is a commonly recurring fault, advice over the phone may fix it. Most of the time, however, the symptoms are insufficient to diagnose; so, the first step is to run the daily tests. If BITE does not disclose more information, then the weekly tests are run. When a fault is indicated, an adjustment or replacement as called for by the Fault Identification Procedures (as modified by personal experience) is made. If replacement of individual modules makes no headway in eliminating the fault, then larger functional assemblies are replaced. When this strategy has finally brought HIPIR back to correct function, then replaced components are carefully swapped back in until it fails again. There is some risk to this final operation, since it can cause other failures indirectly.

II. BACKGROUND ON THE MATIE PROJECT

This report describes the results of the Maintainer's Associate Training Instructional Environment (MATIE) project. This effort is the latest in over 20 years of research and development (R&D) in the field of maintenance training conducted by the AFHRL. The specific line of R&D dealing with maintenance training for the F-111 Avionics Intermediate Shop 6883 Test Station, the object of training in MATIE, began in the late 1970s with a cost and training effectiveness study of a three-dimensional simulator versus an actual equipment trainer (Cicchinelli, Harmon, Keller, & Kottenstette, 1980). Following this, a two-dimensional, flat panel simulator was developed and subjected to a similar comparison (Cicchinelli, Harmon, & Keller, 1982). After this, the Interactive Graphics Simulator project (Pieper et al., 1984) studied the viability of computer-based troubleshooting simulations with videodisc and computer-generated imagery. In the graphics simulator study, the focus of simulation was shifted from the equipment and physical fidelity to the troubleshooting task and cognitive fidelity. In so doing, much of the groundwork was laid for building an expert-system-based job aid for 6883 maintainers. Furthermore, such an expert system, it was thought, would serve as the expert module of an intelligent tutoring system, MATIE.

AFHRL research in artificial intelligence applications in maintenance training began in the mid-1970s with the SOPHIE project (Brown et al., 1982). Since 1982 and the publication of a report analyzing the potential of AI for training, performance measurement, and job performance aiding (Richardson, 1983), AI applications to training have been a priority at the Laboratory. In 1984, the Training Systems Division began investigations of expert systems for equipment maintenance as a prerequisite to building ITSs for maintenance. This work culminated in 1985 with a field demonstration of an intelligent maintenance aid, called the Maintainer's Associate, for the 6883 test station (Richardson, Anselme, Harmon, Keller, & Moul, 1986).

The Maintainer's Associate system used a symptom-based approach to diagnosis. A fault tree was developed by knowledge engineers. The principal basis of this fault tree was a high- level block diagram of the system under test, synthesized by the knowledge engineers from technical data and conversations with SMEs. This diagram did not exist prior to this effort. As a high-level diagram, the complexity of detail of the 6883 test station was abstracted away. This was possible because an automatic test station is essentially a huge switching network. The general switching task always remained the same (routing a stimulus signal through a unit under test to a response measurement device for comparison to a standard). Thus, the high-level block diagram represented the general switching task. With the diagram serving as a device model, the knowledge engineers worked with SMEs to develop the preferred test strategy, a fault tree, for the 6883 test station. This test strategy served as the control strategy for the diagnostic problem solving, providing the focus of attention at the abstract, block diagram level.

But to serve as a job aid, real tests had to be made, not abstract ones. Real wires in real physical locations carrying specific, real signals had to be checked, not abstract functional relationships.

The detailed configuration of the test station was determined by test programming associated with a test number from the sequence of tests written to exercise the unit under test. In order to know the specific signal value and location associated with a connection shown as a single line on the general block diagram, a set of tables was developed that related for each given test number the specific signal value and location active for each region of the block diagram. To illustrate, while the block diagram showed only one connection from the Switching Control Unit to the Test Point Relays, in reality there were hundreds of wires linking these two functional areas. The tables, indexed by test number, were used to look up, for a given test number, which one or two of those hundreds of wires was active and what signal it was carrying.

The tables were developed by a parser which took as input the test station programming (by test number) and technical data (obtained from tables in the Technical Orders) and provided as output tables of signal locations and values for all block diagram regions by test number. The parser was written as computer code, not as an expert system. The four-step process involved was not obvious in the code, nor was information about which tables were used or how they were linked.

This test tree was represented and interpreted in an expert system shell developed by General Dynamics--RuleKit. RuleKit executed an interpretive cycle implementing the "establish and refine" problem-solving methodology (Chandrasekaran, 1983; Tanner & Bylander, 1984). In this strategy, a probable cause of failure is "refined" into a set of more finely discriminated probable causes. Then, evidence is gathered for and against each of these until one is "established"; that is, until sufficient evidence is gathered that indicates that it is now the probable source of the malfunction. This process is repeated until the fault is refined to a sufficiently fine level of detail--a "black box," a card, or a piece part, depending on the circumstances.

The test tree indicated which abstract region to test, and then given the test number at which a failure occurred, a table look-up provided the exact signal location and expected value.

MATIE was designed to build upon the expert system developed for the Maintainer's Associate. The approach was to use the same fault tree and establish/refine methodology. However, the user interface was to be "inverted," exposing the user to the establish/refine interpretive cycle. That is, instead of the system refining a probable cause into a set of probable causes, the user would be asked to do this. Then instead of the system asking the most pertinent question to establish one probable cause as the culprit, the user would be asked to propose the most relevant question. This inversion was to be augmented with a student modeling module and a means of selecting problems, a curriculum module. Focus was to be on the test tree associated with the abstract block diagram. Students could be asked to provide signal locations and values, and their answers would be compared to those provided by the table look-up.

MATIE Research Questions

The MATIE project posed two main research questions. The first involved a reexamination of the issues investigated by Clancey and Letsinger (1981) in adapting a previously constructed expert system to the role of expert module in an ITS. The principal concern, raised by adverse experience with MYCIN, was that an expert system designed as a consultation system or job aid might not be suitable for the purposes of instruction if the rules involved did not explicitly represent the deep and fundamental reasoning and knowledge involved in the problem solving. This deep knowledge seems necessary to provide instructive, memorable, and theoretically motivated explanations to students. Much of MYCIN's knowledge base was found to be lacking in this more principled representation of the task, with the knowledge lying tacit in the surface structure of the rules.

A second research issue of concern in the MATIE project was the possibility of creating an ITS development environment, or shell (*ATIE). Would it be possible to construct MATIE in such a way that the tutor could easily be extended to other diagnostic problems, or even to other domains?

III. APPROACH

The approach taken in the original design of MATIE (Jones, Bamford, Richardson, & Sizer, 1987) sought to preserve the expert system previously developed for the 6883's Maintainer's Associate (Richardson et al., 1986). The idea was to "invert" the role of computer and user in such a way that instead of providing direction for each diagnostic step, the system would now quiz the user (i.e., trainee) for the next most appropriate test point. MATIE would itself be able to determine this next step by utilizing the same diagnostic tree upon which the expert system for the Maintainer's Associate was based. Augmenting this inversion with a "student modeling module" and an "instructional module" would allow students to be tutored according to the adequacy of their responses as compared to those of the expert module. Figure 3 illustrates the concept of this type of inversion.

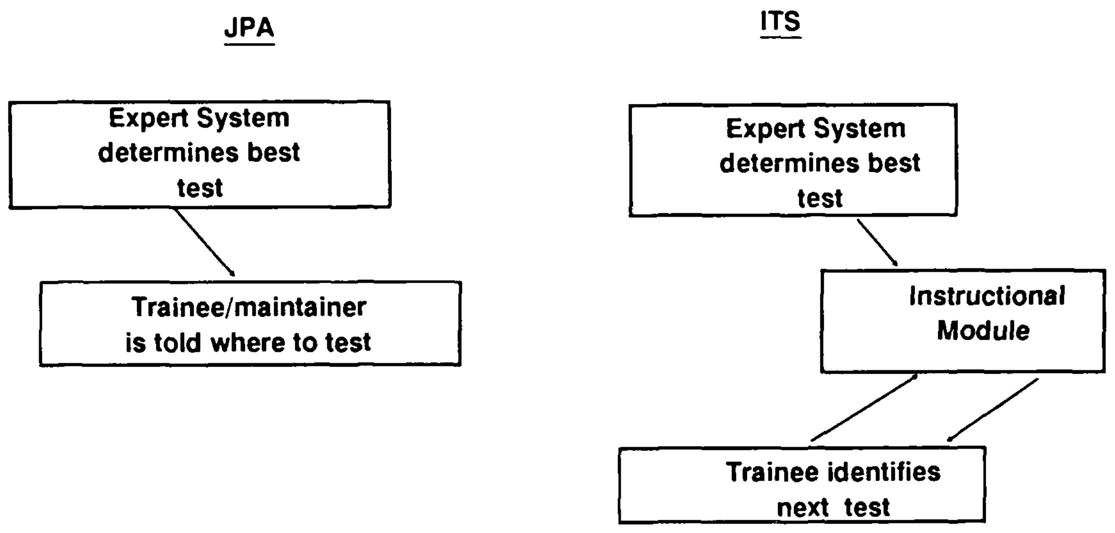

JPA
ITS
Expert System determines best test
Trainee/maintainer is told where to test
Expert System determines best test
Instructional Module
Trainee identifies next test

Figure 3. **The concept of inversion of an expert system for use in an ITS.**

It was felt that the establish/refine problem-solving strategy was a typical and relevant methodology for students to learn. Further, it was felt that since this strategy was explicit in the interpretive cycle of the Maintainer's Associate expert module, the problems engendered with implicit control knowledge (Clancey, 1986) would be avoided. Also, it was believed that the 6883-specific fault tree interpreted by RuleKit represented the proper troubleshooting strategy for this equipment. Student use of the technical publications needed for deriving signal locations and their values could be tested by asking for these data and comparing them to the look-up table.

One advantage of this approach was that MATIE would be able to make use of an expert system already in existence. The considerable expense involved in this facet of the ITS development process would thus be obviated. Moreover, once a basic "shell" for MATIE had been constructed (i.e., incorporation of the student and instructional modules), it should be possible to build future ITSs for other troubleshooting problems (or perhaps even for other domains) at a minimal cost, simply by substituting a new expert system module into the generic shell (*ATIE). While Jones et al. (1987) point out that complete modularity would not be entirely possible, the reduced cost of using an *ATIE in further development efforts represents a significant benefit. This "generic" aspect was, in fact, a secondary objective of the original MATIE design.

Problems with the Inversion Approach

Unfortunately, the educational implications of developing an ITS based on the Maintainer's Associate expert system interfere with MATIE's primary objective of teaching diagnostic problem- solving skills (i.e., the five-step test-refine cycle defined by Richardson, Keller, Maxion, Polson, & DeJong, 1985, p. 36), because the former expert system module was designed as a precompiled tree structure. Although the diagnostic tree was itself derived from the human troubleshooting endeavor, its implementation did not closely model that process. That is, rather than having to determine the remaining best test point upon each iteration of the diagnostic cycle (based on the implications of both "good" and "bad" results for any of the remaining test candidates), the Maintainer's Associate simply followed a branch of the test tree to the next "child" node. Both of these processes yield further refinements to increasingly smaller 6883 subcomponents, until the problem "Test Replaceable Unit" (TRU) is finally isolated. But the "tree" method uses *a priori*, "precompiled" knowledge, whereas the practice with humans involves making such decisions in "real time." Therefore, if the tree were used as the basis for teaching electronics technicians about fault diagnosis, the principal subject matter would involve only superficial knowledge about the tree structure itself (i.e., where to test next, given the results and implications of previous tests). The key problem-solving activity of identifying the next optimum test, after having first considered the implications of using any of the remaining test points (in addition to the results and implications of past tests), would be excluded.

Besides failing to teach this critical feature of the test-refine cycle, a second difficulty with employing the Maintainer's Associate expert system for MATIE has to do with the fundamentally "static" nature of the expert system. Because all testing decisions were predetermined in the Maintainer's Associate design phase, and because optimal search paths were at that time most important, only single-choice alternatives were given at each step of the troubleshooting process. By necessity, some of these selections were made arbitrarily. As a result, even when alternative test points yield equivalent implications, the tree is too inflexible to accept their choice. Thus, while problem solving in real time might legitimately select any of several equivalent test points, teaching only the structure of the diagnostic tree would not permit such alternatives.

Another byproduct of using precompiled test logic in the Maintainer's Associate expert system was that access to a rationale for selecting particular test points was not available for later access. Key problem-solving activities were carried out by knowledge engineers and SMEs during the design phase, with only the result of their analyses being conveyed by the expert system's rule base. As a result, crucial "why" knowledge was unavailable for subsequent articulation by an ITS: that is, though the Maintainer's Associate could make all of the proper test point selections, its expert system did not provide for later explanation of those decisions. This would constitute a significant departure from previous ITSs based upon expert system inversions, such as GUIDON (Clancey, 1987), and significantly handicap MATIE's instructional module in its ability to explain its various decisions.

Facilitating the Inversion: A Solution

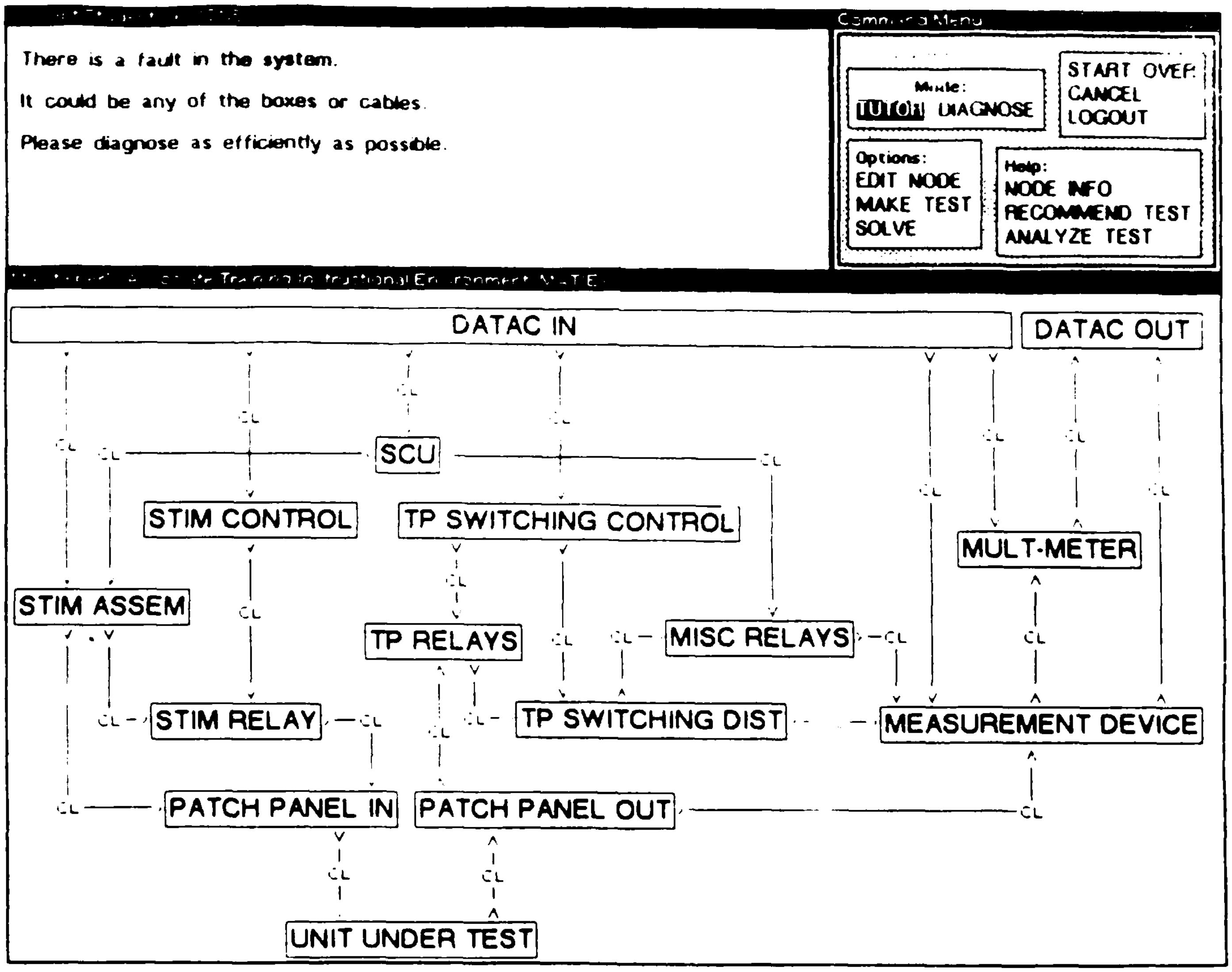
For the above reasons, it was deemed impossible to invert the expert system of the Maintainer's Associate in its current state. Thus, a portion of the development effort for MATIE involved the redesign and construction of an appropriate expert system module. Because the ITS needed to closely parallel the troubleshooting behavior of human experts (i.e., attain a high degree of cognitive fidelity), a model-based approach was adopted. This permitted the ITS to anticipate the diagnostic problem-solving steps its human trainees were to conduct upon each iteration of the test-refine cycle. By using an expert module that simulated the human activities involved in diagnostic problem-solving, it was possible to invert the role of computer and user in the manner intended, while also preserving MATIE's stated educational objectives.

It was decided to view the test station at the level of abstraction represented by the 6883's block diagram, independent of test number. Knowledge of the test number was typically needed only to figure out signal value and location; that is, to instantiate a block diagram connection to the physical location and signal value applicable for this connection and test number. The block diagram represented that aspect of the troubleshooting problem which is invariant with respect to test number. This perspective provides the overall control structure for the diagnostic strategy appropriate to troubleshooting automatic test equipment. Doing actual tests, of course, would involve the additional step of instantiation. But this step was not seen by instructors as an instructional challenge. For them, troubleshooting the block diagram was the primary objective.

Representing the problem at a level of abstraction which was invariant with respect to test number also dictated that signal value and location would not be incorporated in MATIE's design. Instructionally, this task was not viewed by trainers as an objective. Technical publications usage and the details of the 6883 programming were not seen to present difficulties to students. But even if determining signal value and locations were to have been tutored, the Maintainer's Associate did not support skill acquisition in this area. All that could be done was to judge a student's answer as to signal value and location as correct or incorrect. No help could be provided because the knowledge upon which that instruction would be based was locked up tightly in the computer code of the parser that had originally generated the tables.

MATIE Design Specifications

MATIE incorporates the conventional features of an ITS--an expert module, a student modeling module, and an instructional module--within the context of a user interface that permits students and instructors to "logon" and access various system routines. The following sections describe in detail the function of eaci of these components. Throughout this discussion it may prove helpful to refer to MATIE's main display, a graphic representation of the 6883 test station's topology, as shown in Figure 4.


There is a fault in the system.

It could be any of the boxes or cables.

Please diagnose as efficiently as possible.

Figure 4. **MATIE's Graphic Representation of the 6883 Topology.**

Expert Module. The expert module implemented for MATIE was significantly altered from that used previously by the 6883's Maintainer's Associate. The former was a rule-based expert system: probable-cause rules were used to further refine the troubleshooting task at each level of the diagnostic hierarchy. The model-based approach adopted for MATIE, however, did not include the conventional expert system components (a rule-base and inference engine). As such, it does not qualify as an "expert system" *per se*, although it satisfies the functional requirement of "solving problems as would human experts." This latter point was a crucial feature of the new expert module, since it was necessary that MATIE's expert module parallel the human problem-solving endeavor in order to allow meaningful tutorial interactions.

While the redesign of MATIE's expert module constituted a significant portion of the development effort, an important benefit of the Maintainer's Associate project was that the knowledge engineering task did not have to be duplicated. The Maintainer's Associate authors had made painstaking efforts to accurately characterize the knowledge of troubleshooting experts. Four SMEs were employed, two representing a training perspective and two representing the field technician's viewpoint. Together with the knowledge engineers, and closely adhering to available technical documentation, a TRU-level, specification-based representation of the 6883 troubleshooting task was generated. This essentially constituted a "device model"--a symbolic map of individual device components, along with their input/output behavior and interconnections. It was concluded that this characterization accurately represented the 6883 test station, both topologically and teleologically. Because of the care given in this first effort, it was possible to use this result in redesigning the expert module, with the diagnostic task now being conducted in real-time.

The diagnostic task itself is essentially a process of hypothesis refinement. After the fault is isolated to a set of probable causes, a refined subset can be produced on the basis of a test result. Further testing will eventually reduce the probable-cause set to a single member: the solution. As defined by Richardson et al. (1985, p. 36), this process consists of five steps, which are performed iteratively:

1. Decide whether further diagnostic refinement is warranted.

2. Select where to measure next, such that expected information gain per unit cost is maximized.

3. Identify the expected (correctly functioning) value of the selected measurement.

4. Make the measurement.

5. Determine the implication of this measurement in terms of component blame or innocence.

Each iteration begins with Step 1; the technician proceeds with the remaining steps if the solution has not yet been determined. Step 2 is most critical to the diagnostic process; it is this aspect that MATIE's expert module seeks to emulate (and which prior systems have failed to model). Steps 3 and 4 are in some sense trivial to the present instructional objectives, since MATIE was not intended to address particular test procedures or measurement values. Step 5 follows through with the implications of one of the potential tests considered earlier in Step 2, eliminating some members of the probable-cause set and leaving behind a refined subset. The entire process then continues by returning to Step 1.

Adequate performance of the second step is the key to efficient 6883 troubleshooting. The objective here is to consider the possible refinements that would result from selecting any of the remaining test points (arrowheads, in Figure 4) in the set of probable causes. For each of these, estimates are made as to the number of TRUs (labeled boxes and connecting lines, in Figure 4) that would be eliminated for both a GOOD test result and for a BAD test result. A GOOD result means that the tested result matched the expected result, and indicates that TRUs in the path leading up to that point were not problematic (i.e., the fault must lie somewhere beyond that point). A BAD test result means that some discrepancy between the tested and expected result was found, and that the fault must therefore lie along the path leading up to that point (i.e., TRUs beyond that point can be eliminated from further consideration). Test points which yield the maximum potential information gain (i.e., greatest refinement of the probable-cause set) are the most likely candidates for the next test.

This turns out to be a rather sophisticated task. Information gain is greatest when the sum of TRUs eliminated from further consideration for both a GOOD and a BAD test result is maximal, and when the difference between the number of TRUs eliminated by GOOD and BAD tests is minimal. For example, a test point that eliminates five TRUs if the test result is GOOD and five TRUs if the result is BAD would be preferable to another test point that eliminates four TRUs and six TRUs for GOOD and BAD test results, respectively. Both of these, of course, would be preferred to a test point that yields a combination of seven and three TRUs eliminated--or worse, six and three, or five and two, etc. (some combination less than ten). Furthermore, consideration must be given to the costs involved in making each test (time, effort, dollars, etc.) and the probability that the tested TRUs are likely to be faulty (according to component life-cycle expectations relative to time since last replacement). It should be appreciated that when this process must be repeated for each of the remaining test points (or at least those that would eliminate some non-trivial number of TRUs), the diagnostic task is certainly not a simple matter.

It was precisely these features that MATIE's expert module sought to emulate. They are represented in Steps (B) and (C) of the following algorithm:

A. Define and redisplay the graph's current state, if still unsolved.

B. Analyze the remaining SUSPECT Test Points (TPs):

```
FOR each remaining SUSPECT Test Point (TP) DO BEGIN
  IF the test result is "GOOD" THEN
    A = The number of nodes that would be eliminated
    B = Sum of the failure rates of SUSPECTs not eliminated
    Temp(GOOD) = A * B
  ELSE (the test result is BAD)
    A = The number of nodes that would be eliminated
    B = Sum of the failure rates of SUSPECTs not eliminated
    Temp(BAD) = A * B
  Rating(TP) = [Temp(GOOD) + Temp(BAD)] /TP cost
END (loop for SUSPECT analysis)
```

C. Rank-order these Test Points according to their Ratings.

Step (A) corresponds to the issues addressed by Steps 1 and 5 in the human-expert troubleshooting algorithm. When choosing the first test point, Step 5 is irrelevant, and Step 1 is trivial. But for every succeeding iteration of the test-refine cycle, the current state of the diagnosis must be understood or stated by humans, and redefined/redisplayed by MATIE. Test points (TPs) may be classified as either GOOD or CLEAR if prior tests have removed their corresponding TRUs from the probable-cause set, and BAD if test results have allowed them to remain SUSPECT (i.e., potentially faulty). The distinction between GOOD and CLEAR TRUs is subtle: GOOD TRUs are those which have been eliminated from further consideration because the source of the signal for a particular test passed directly through them, while CLEAR TRUs are those which have been eliminated because their placement in the 6883 architecture could not have contributed to a BAD test result (this is an example of one of the finer discriminations that expert troubleshooters are required to make in their diagnostic problem-solving).

Step (B) carries out the human-expert algorithm's important Step 2 considerations. This is accomplished by evaluating each of the remaining SUSPECT TPs in terms of its neighboring TRUs. Ratings are calculated for both GOOD and BAD test results. For each possibility this involves (a) determining the number of TRUs that would be eliminated from further consideration (i.e., judged to be either GOOD or CLEAR), and (b) estimating the probability that the TP would indeed give that GOOD or BAD test result (by summing the failure rates for the TRUs it would allow to remain SUSPECT). The product of these sums then constitutes a "rating" for either the GOOD or the BAD test result for that particular TP. Finally, a composite "Rating" is computed by dividing the cost of that test into the sum of the GOOD and BAD TP ratings.

An example may help to illustrate this process. Suppose a TP eliminates three of ten remaining TRUs if the test result is GOOD, and seven TRUs if the result is BAD. The GOOD rating would then be the product of 3 times the sum of the failure rates for the other seven TRUs. If this latter figure was, say, 5.0, then the GOOD rating would be 3 x 5.0, or 15.0. On the other hand, the BAD rating would be the product of 7 times the sum of the failure rates for the other three TRUs, say 3.0. This would yield a rating of 21.0 for the BAD test result. The final rating for this TP, then, would be 15.0 + 21.0 divided by the cost of running the test. If the cost was 2.0, for instance, the final rating for this particular TP would be 18.0.

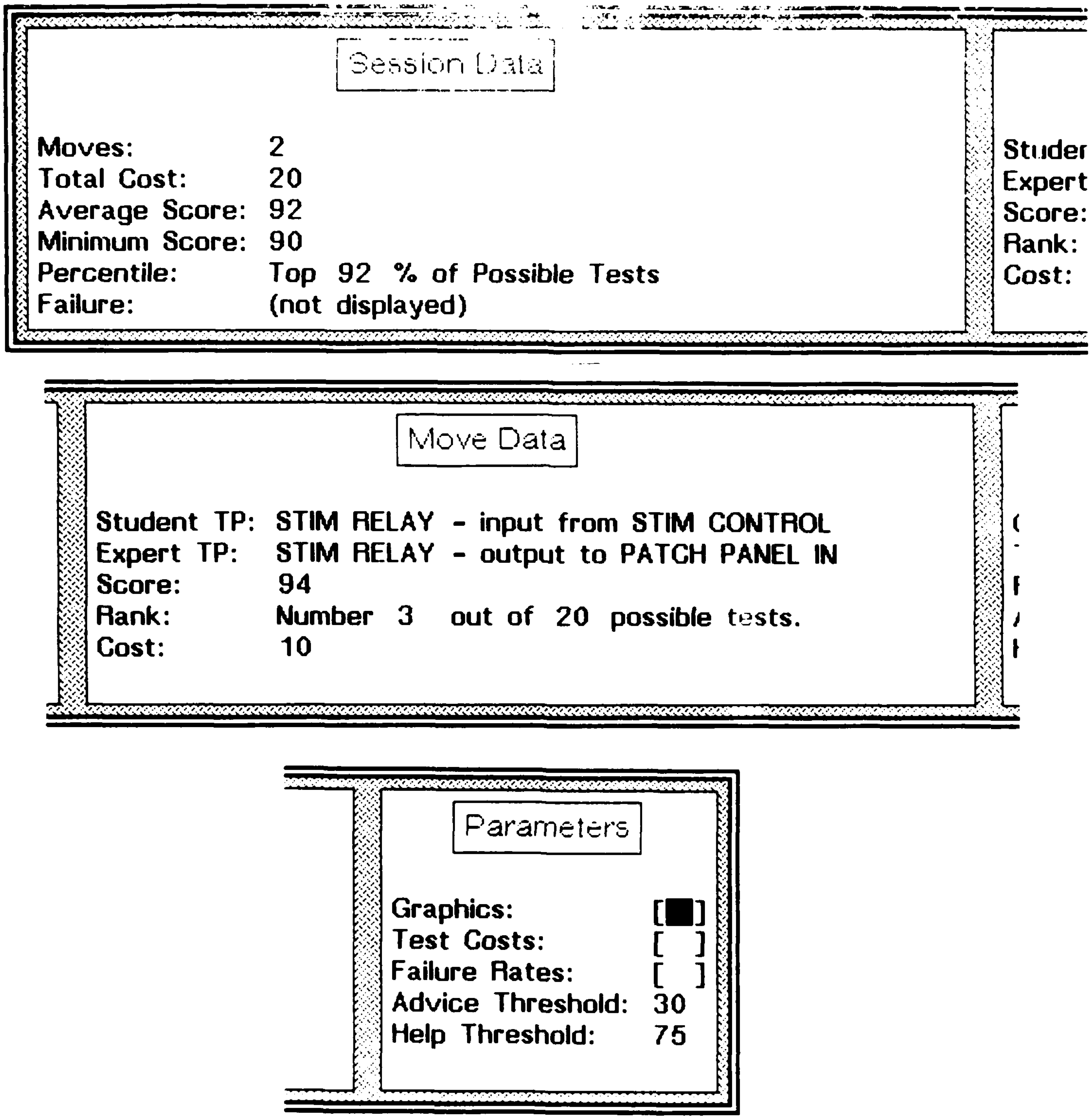
After a rating has been computed for each of the remaining SUSPECT TPs, Step (C) rank-orders them from high ratings to low. This gives a list of TPs, ordered according to their preference for testing. If two or more TPs should happen to have equivalent ratings, they are equally preferred to any lesser ratings. Thus, following our example from above, if the other nine TPs yielded ratings of 16.0, 12.5, 12.0, 8.0, 7.5, 6.0, 4.5, 3.0, and 2.0, then the TP evaluated previously (with a rating of 18.0) would be that which MATIE's expert module judged to be most beneficial at this particular point in the diagnostic sequence.

Student Module. It would be inaccurate to call the present Student Module a "model" of student troubleshooting skills (VanLehn, 1988). In its present state, it might be better labeled a student evaluator or profiler because it is now limited to tracking basic statistical information regarding student performance on individual tests (moves) and over the entire lesson (session). This differs from the two most common ITS methods for modeling student skills: overlaying upon an expert-system-based student model when mastery of its components has been demonstrated, or marking weak areas (bugs) in such a model when performance errors are detected. Figure 5 presents the MATIE window image that provides this information to the trainee.

The "Session Data" area provides summary information on the accumulated "move" data up to the current point in the lesson. This includes the total number of tests made, the summed "cost" of those tests, the average (arithmetic mean) of the student's scores, and the minimum (worst) score attained to that point. "Percentile" refers to the averaged rank of the tests made to this point, and is expressed as a percentile (i.e., 1 minus the average percentage). Thus, if only one test had been made, and its rank was 4 out of 20, the percentile figure would be 80. If the next test yielded a rank of 4 out of 10 (a percentile of 60), the averaged percentile at this point would be 70 (i.e., [80 + 60] / 2). A final datum in this display, "Failure," is presented only to instructors; it indicates which TRU is faulty for the current session.

The "Move Data" area gives information on the student's most recent selection. The TP chosen by the student and the one at the head of MATIE's rank-ordered list are the first two items displayed. Next comes the score of the student's TP, computed by dividing the rating of the student-selected TP by that of the expert module. "Rank" indicates the position of the student's selection in the expert module's rank- ordered list, and is expressed as a ratio (to the number of remaining TPs). "Cost" refers to the expense (time, effort, dollars, etc.) involved in making the test.

Instructional Environment. The "instructional environment" refers to the context in which tutoring occurs. MATIE's principal feature is its graphic representation of the 6883's topology. Students interact with this environment by evaluating and choosing TPs, and by observing the implications of those tests in terms of their refinement of the probable-cause set. MATIE provides several "Parameters" (see Figure 5) that can assist students in these activities. The manner in which these functions operate is established by the trainee's instructor, by way of the Instructor Interface (described below).



```
Session Data

Moves:           2
Total Cost:      20
Average Score:   92
Minimum Score:   90
Percentile:      Top 92 % of Possible Tests
Failure:         (not displayed)
```

```
Move Data

Student TP:   STIM RELAY - input from STIM CONTROL
Expert TP:    STIM RELAY - output to PATCH PANEL IN
Score:        94
Rank:         Number 3  out of 20 possible tests.
Cost:         10
```

```
Parameters

Graphics:            [■]
Test Costs:          [ ]
Failure Rates:       [ ]
Advice Threshold:    30
Help Threshold:      75
```

Figure 5. **MATIE's Display of Current-User Data.**

The first three parameters are controlled by binary switches; they can be set either to "ON" (in which case they are active), or to "OFF." "Graphics" refers to a highlighting (reverse-video) capability for TRUs that have been evaluated as GOOD or CLEAR by prior test results (i.e., they are no longer SUSPECT). When "ON," TRUs that have already been eliminated are darkened; those remaining in the probable-cause set continue to be shown as in the original presentation (Figure 4). This switch also affects the display for the "Analyze Test" function (see below): TRUs eliminated by a GOOD test result will blink on and off, and are followed by a similar display for those eliminated by a BAD result.

The other two binary switches, "Test Costs" and 'Failure Rates," determine whether these kinds of data are taken into account by the expert module when TP scores are computed. This will not happen if the appropriate switch is set to "OFF." If neither switch is "ON," scores will only be figured according to the number of TRUs that can potentially be eliminated. These two parameters will also affect the amount of information displayed in the "Node Info" display. Test cost and failure rate data are shown only if they are currently active (i.e., if they affect the expert module's determination of the TP list).

The final two parameters are controlled by threshold values rather than binary switches. "Advice" refers to the activation of MATIE's coaching function: TP selections having a score less than or equal to the preset threshold will not be allowed, and coaching will instead be provided. The nature of this advice is explained in greater detail in the section on the Instruction Module (below).

The "Help" parameter determines when students can activate the second two of MATIE's three assistance functions, which are presented in the menu of Figure 4. The first, "Node Info," is always available to the student. But the others become active only when the average score for a lesson equals or falls below a threshold previously established by the instructor. Help can be accessed on the following topics:

1. Node Info: This function permits the student to access the current status for a selected TP, as well as its test cost and TRU failure rate data.

2. Analyze Test: For a selected TP, data will be provided on the number of TRUs eliminated for both GOOD and BAD test results (with optional "graphics" for each), the cost of that test, its score and its rank.

3. Recommend Test: This function will highlight MATIE's preferred TP (i.e., that which is ranked first in the expert module's probable-cause list).

If either of the latter two "Help" functions is accessed when the current session's average score is greater than the preset threshold, the following message will instead be displayed: "You are doing quite well, actually. Try to do it without my help."

In addition to these Parameters and Help functions, Figure 4 shows two operational "Modes" and three user "Options" (one of them available for instructors only). The provided "Options" indicate the kinds of actions students can take when using the system. All moves except the last are "Make Test" actions: The student's task is to select which TP to test next. When the graph has been completely analyzed, leaving a final TRU, a "Solve" is conducted. This is the point at which the student indicates which TRU is faulty. Any "Solve" made prior to the graph's ultimate refinement constitutes a "guess." Either of these two options must be selected before a move can be made. The first Option, "Edit Node," can be accessed only by instructors. It permits modification of the "test cost" and "failure rate" data for any selected TP, box, or cable.

MATIE functions as the ITS described above when in "Tutor" Mode, but permits more exploratory interactions when set to "Diagnose." All of the above characteristics still apply, but when a "Make Test" Option is undertaken, the user is requested to indicate whether the test result is GOOD or BAD. This permits examination of the graph without regard to a background lesson and faulted TRU. The student can thus attempt various tests and determine their implications for neighboring nodes. In essence, MATIE functions under these conditions as did the Maintainer's Associate developed by Richardson et al. (1986), with the user indicating outcomes for various test results and MATIE suggesting the next best TP.

Curriculum Module. The curriculum module was implemented as originally designed by Jones et al. (1987). Each lesson is based on the location of a fault from among the available TRUs. The curriculum data base thus consists of a lesson for each TRU, with the results for each of its TPs (GOOD or BAD) so labeled. (The construction of lessons is itself explained below in the section on the Instructor Interface.)

Troubleshooting remains invariant across lessons; students perform the same diagnostic task, but with a different faulty TRU in each. Because the sequencing of these lessons makes little difference to the development of an efficient algorithm, they are presented to the student in random order.

The diagnostic task increases in difficulty as students are asked to perform without the aid of graphics (to indicate already eliminated TRUs), or to take into account test costs and failure rates. Therefore, the Parameters previously discussed (under the "Instructional Environment" heading) represent the best means for instructors to facilitate an increase in the sophistication of student performance. Adjusting the two threshold parameters represents a means of change within a variety of lessons.

For instance, students might initially be asked to master lessons without test costs or failure rates, and with "graphics" enabled. As performance improved, however, threshold values might gradually be raised until Help became unavailable; mastery would at this point be demonstrated. A next step might be to remove graphics, or to include one of the two TP parameters, with thresholds again lowered. After achieving this level of difficulty, a further requirement could be added. Finally, the student would have mastered troubleshooting in a graph that had close physical similarity to the 6883 test station itself. These concerns are a matter for further study, however; for now, they have been left to the instructor's discretion.

Instruction Module. The instruction module fills the role traditionally held by the teacher or expert tutor. Its principal function is to offer selective advice to students as they proceed through a lesson. As explained in the section on Instructional Environment, MATIE activates this coaching only when the score for the student's current move falls below the instructor-chosen advice threshold. The first action taken is to prevent the selection of the indicated TP, and to suggest that the student try another: "That's an unacceptable test point! Please come up with a better one than that." If the next choice also falls below threshold, this same presentation is made along with the "Analyze Test" Help function--"Would you like to see why?"--to which the user confirms by clicking on "Yes" or "No." A third error will produce the same two levels of advice and add the "Recommend Test" Help function--"Would you like to see which one I would choose?"--plus the "Yes" or "No" request. Thus, TP choices below the indicated threshold are never permitted, and up to three levels of advice can be offered, depending on the number of successive "bad" choices made.

Instructor Interface. When students log onto the system, they immediately proceed to MATIE's 6883 display (Figure 4) and begin work on a lesson. For the instructor, however, a student roster and menu of actions is first displayed (Figure 6).

The first four menu items pertain to student management activities. Instructors can add students to the roster, change the spelling of student names, delete students from the class, or modify the Parameters under which a student's lessons are run (per the Instructional Environment). Each of these operations is fully prompted by user messages.

The next two menu items will collect statistics or display histograms for as many students whose names have been selected on the roster. The three mouse buttons are differentiated for this purpose: The left button will select single students only; the middle button will select multiple students (i.e., those already selected are not de-selected when the middle button is used for clicking); and the right button can be used to select a range of students (all those between the one(s) already chosen and the one for whom the right button is clicked will be picked). Statistics (Figure 7) will be displayed for as many students as have been selected; if data are shown for more than one student, then a row of averaged data will also be presented at the bottom of the display. Histograms, if requested for one student, will display the session scores for each lesson, as well as an average score (Figure 8). If multiple students are selected, then each student's average score is displayed, along with a class average (Figure 9).

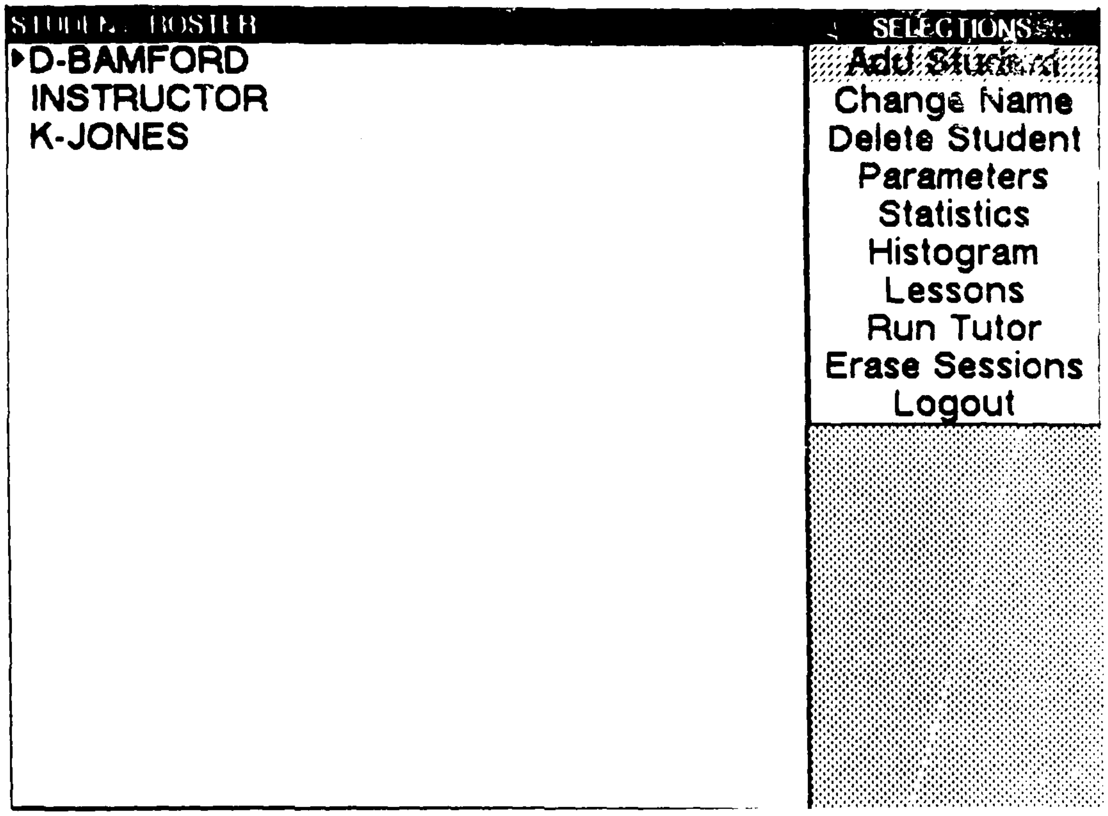
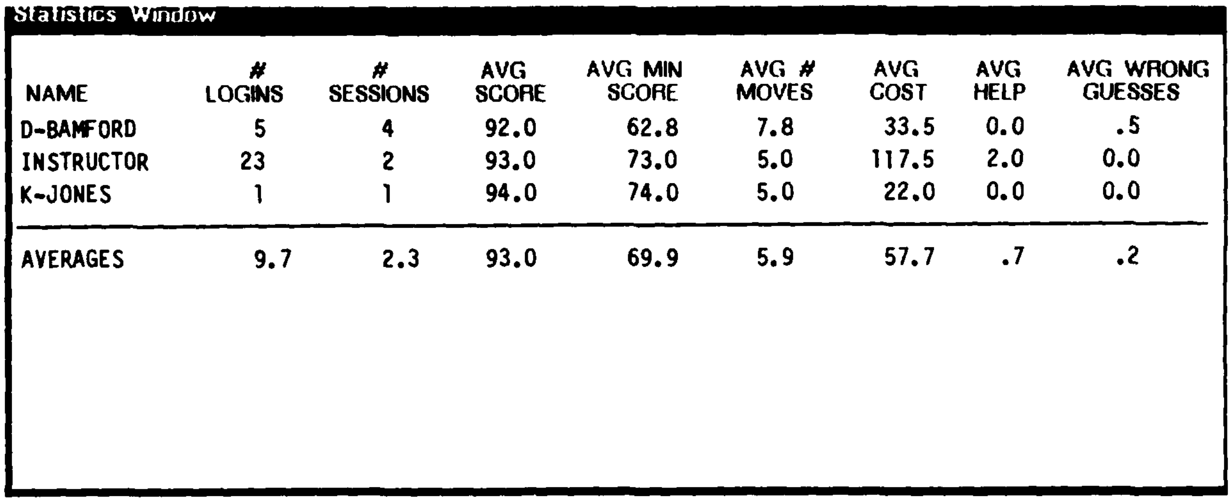


Figure 6. MATIE's Instructor Interface Roster and Menu.


Statistics Window

NAME	# LOGINS	# SESSIONS	AVG SCORE	AVG MIN SCORE	AVG # MOVES	AVG COST	AVG HELP	AVG WRONG GUESSES
D-BAMFORD	5	4	92.0	62.8	7.8	33.5	0.0	.5
INSTRUCTOR	23	2	93.0	73.0	5.0	117.5	2.0	0.0
K-JONES	1	1	94.0	74.0	5.0	22.0	0.0	0.0
AVERAGES	9.7	2.3	93.0	69.9	5.9	57.7	.7	.2

Figure 7. MATIE's Display of User Statistics.

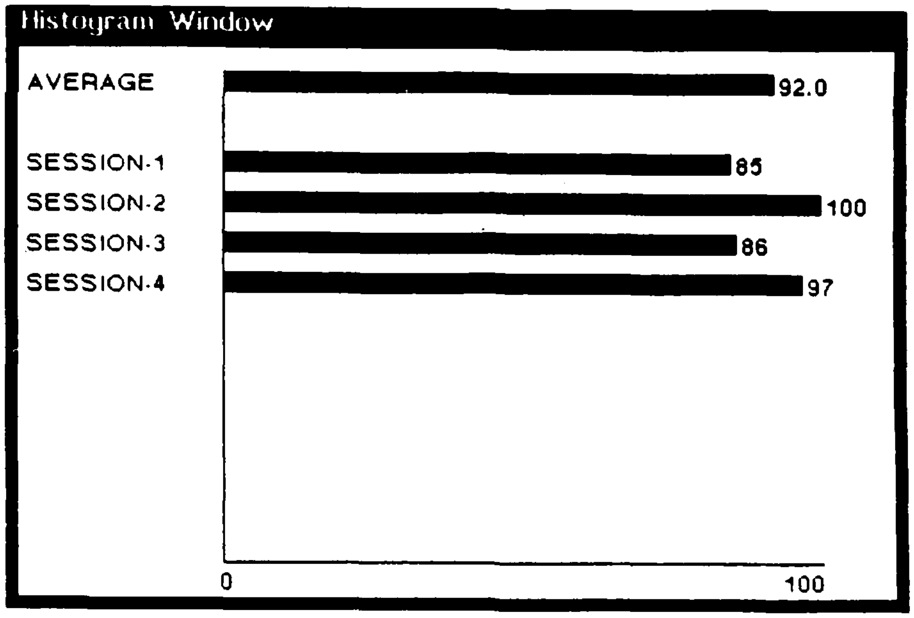


Figure 8. **MATIE's Histogram Display for a Single User.**

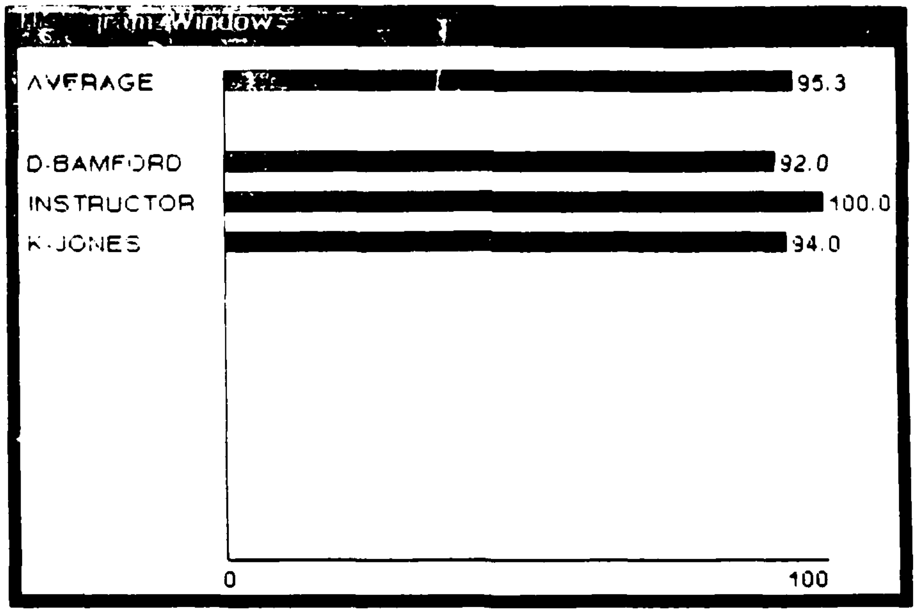
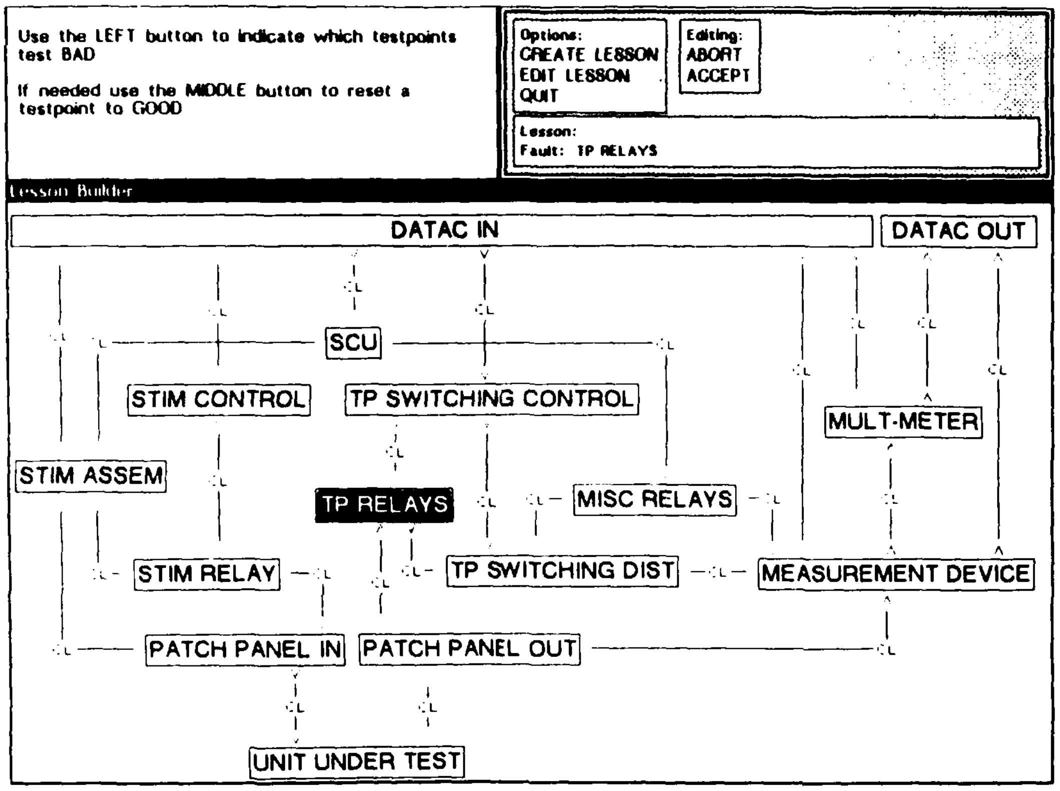


Figure 9. **MATIE's Histogram Display for Multiple Users.**

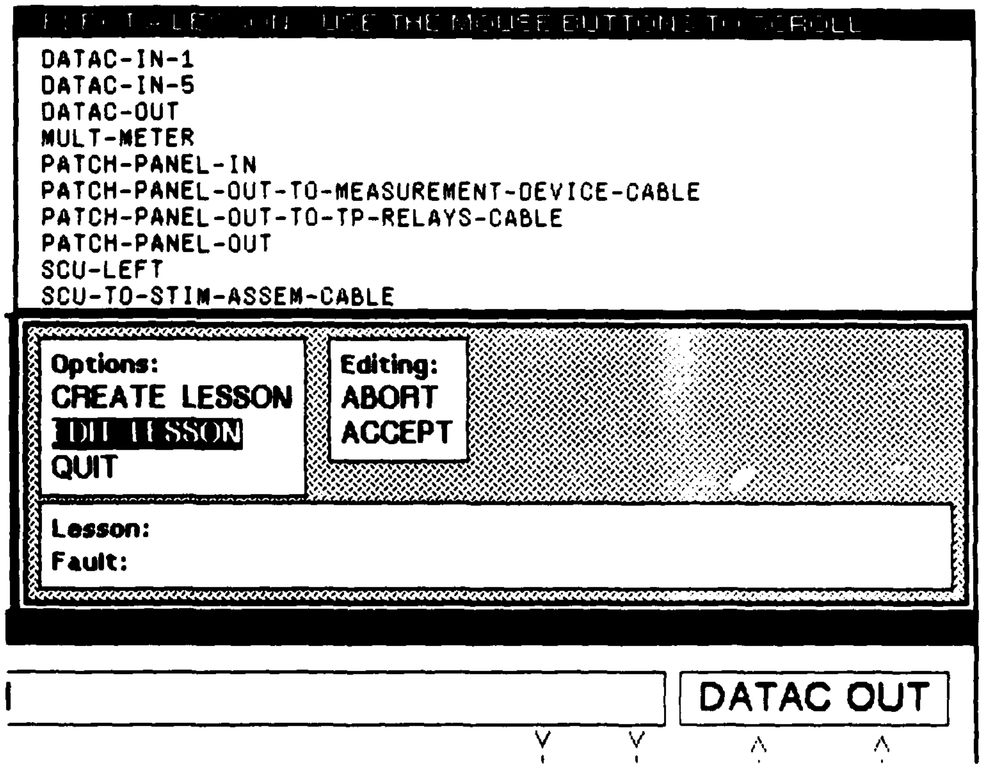
Through the "Lessons" option, instructors can also create new lessons or edit previously created lessons that are on file. After entering this mode (Figure 10), the first step is to pick which "Option" to use. If the decision to create a new lesson is made, then the next step is to designate which TRU is going to be faulty. This is highlighted on the graph, and an entry is made in the menu screen under "Fault." The next step is for the instructor to indicate (by clicking the mouse on various TPs) those TPs that would show a BAD result if they were to be tested by the student during a lesson; a provision is also made for correcting BAD TP indications. If the decision to edit existing lessons is made, then a menu is displayed from which a lesson can be chosen (Figure 11); once an item is selected, its present status is displayed on the graph and


Use the LEFT button to indicate which testpoints test BAD

If needed use the MIDDLE button to reset a testpoint to GOOD

Lesson:
Fault: TP RELAYS

Figure 10. **MATIE's Lesson Builder Interface.**


```
DATAC-IN-1
DATAC-IN-5
DATAC-OUT
MULT-METER
PATCH-PANEL-IN
PATCH-PANEL-OUT-TO-MEASUREMENT-DEVICE-CABLE
PATCH-PANEL-OUT-TO-TP-RELAYS-CABLE
PATCH-PANEL-OUT
SCU-LEFT
SCU-TO-STIM-ASSEM-CABLE
```

Lesson:
Fault:

Figure 11. **MATIE's Lesson Editor Menu Display.**

The final three menu selections (from Figure 6) will allow the instructor to run the MATIE tutor (with knowledge available as to the current fault), erase the lesson data for a student (the basis for the statistics and histogram displays), and logout.

IV. CONCLUSIONS

Retrofitting JPAs into ITSs

The first research question dealt with the issues involved in retrofitting an existing expert system to an ITS application. The MATIE project was well informed of this issue, and in its inception, the design contemplated was thought to have avoided problems in this area. The principal way the problem was to be avoided was the use of the establish/refine architecture of RuleKit which modeled explicitly the cognitive steps involved in this type of problem solving.

In spite of this thinking, this approach did fall prey to the "retrofit" issue. If the 6883 rule base had been sufficiently robust in the establish/refine architecture, the approach might have succeeded. However, as discussed, the rule base was degenerate, and did not make meaningful use of the establish/refine interpretive cycle.

Moreover, the probable-cause hierarchy that RuleKit interpreted was provided as a given. But in retrospect, it is seen that the instructional objective of a troubleshooting training system is to help students learn to be able to generate this hierarchy (i.e., a fault tree) from a description of the system under test--from a device model. Since the hierarchy in no way represented the knowledge involved in generating itself, having students learn this hierarchy did not equate to learning model-based diagnostic reasoning. An approach that facilitated the troubleshooting instructional objective was adopted, and the pre-existing approach was abandoned in favor of a model-based approach.

A second instance of the retrofit problem arose in regard to signal location/value instantiation. If this had been an instructional objective, it was observed that it could not have been supported in a tutorial environment given the procedure used to generate these data. The procedure, as noted, was essentially a "black box" expert system, and as such, no interactive, process-oriented guidance could be provided to students on the instantiation task. The only instructional modality available would have been drill and practice, with only "correct/incorrect" feedback available.

In sum, the first lesson with respect to the retrofit question was that, in device diagnosis, the appropriate model of expertise for the purposes of training is the model-based approach. If an expert system implements this approach, it should be useful in an intelligent maintenance training system. The second lesson was that if any relevant problem-solving knowledge is tacit and unavailable (how to build the tree or how to instantiate location/value), the expert system is not suited for the purposes of an expert module in an ITS. This conclusion is softened somewhat by recognition of the existence of the class of ITSs employing the issue-based tutoring method (Polson & Richardson, 1988). That is, it is at times possible to use a black box expert system, if a series of issues can be identified with which to compare and contrast expert and student performance. Indeed, it was a finding of the SOPHIE I project that the black box method could not provide sufficient basis for the kind of instruction, help or guidance necessary to promote improvement in student performance. In contrast, relevant issues upon which to base training are provided by the model-based troubleshooting approach; namely, the details of that algorithm-- propagation and picking out test points which yield maximum information gain per unit cost given tests made already. Thus, the model-based approach is seen to be a necessary precondition for bringing out the issues involved in model-based reasoning.

Extensibility and Generality of MATIE

The second research question dealt with MATIE's extensibility to other troubleshooting problems and to other training domains. The background discussion made the strong case that all troubleshooting tasks are amenable to the model-based approach. Further, the technology (electronic, mechanical, hydraulic, mixed, etc.) of the device under test has no bearing or effect on the preferred model-based approach. Thus, it is concluded that MATIE is extensible to all device troubleshooting problems regardless of technology. This is a sweeping conclusion, one of great generality and potential impact on maintenance training. It is noted that this is the approach employed by other intelligent maintenance training systems, such as IMTS.

MATIE was constructed with an editing capability. The functioning of the tutor is completely driven by the device model. A device model can be rapidly created or edited, including changes in failure rate or test cost. Thus, conversion of MATIE from the 6883 system to other systems is simply a matter of creating a new device model. The training utility of MATIE is device-independent, just as the model-based approach to diagnosis is device-independent. Limitations for MATIE would obtain when signals cannot be effectively represented as "good/bad" or when propagation involves more than simple dependency. That is, in MATIE, each component in the device model has no behavior associated with it. Components simply pass "goodness" or "badness" of input signal along to their outputs. In training cases where more detailed propagation behavior is desired or needed (in which components modify inputs according to behavioral rules or in which components' behavioral rules depend on the state of the components), MATIE would not be appropriate. In sum, MATIE employs a simplified device model (sometimes called a "logic model" (Richardson, 1987), one in which signals are abstracted as "good" or "bad," and one in which components only pass along dependency and do not modify signals. More advanced device-model-based simulators add these capabilities. The IMTS is an example of a more complex device-model simulator. MATIE could be extended to have this capability. MATIE does not do anything differently; it is fundamentally "model-based." It is a simplified version of model-based diagnosis.

As noted, human troubleshooting performance employs a hybrid diagnostic approach, combining symptom-based and model-based reasoning. Thus, troubleshooting trainers should be expected to extend the pure model-based approach of MATIE to the hybrid approach. The case was made that for novices, it is appropriate to limit the training environment to the model-based approach, since this is in accordance with the skill acquisition in this domain. Intermediate or advanced skill-level training in troubleshooting should probably be based on hybrid expert systems, such as IDM or AI-FERRET.

The model-based approach, the second-generation expert system as expert module, is seen to be the appropriate approach for device diagnosis. Can this be extended to other training objectives? Probably so, if the training objective involves a domain which can be modeled as a device or process, containing discrete components or processes which are linked by paths of causality (see Bobrow, 1984). Natural systems may be approximated in this way, to the extent to which we have a mechanistic model for explaining them. But the less mechanistic knowledge available, the more the problem-solving strategy will rely on heuristics, and the less amenable it will be to model-based reasoning. Earthquake prediction, solar flare prediction, medical diagnosis, and weather forecasting all have aspects which severely test our formal, mechanistic knowledge of these phenomena, and thus limit the utility of the MATIE-like approaches to tutoring in these fields. (It is possible that other mechanisms for knowledge representation, such as the adaptive network systems approach discussed in the next section, will help extend the MATIE model to these problem domains.) But by no means should the methodology of model-based reasoning be thought to be limited to device diagnosis. MATIE's approach is probably appropriate far outside the realm of troubleshooting man-made devices.

JPAs Versus ITSs

What light does this effort shed on the relationship and distinction between job performance aids and ITSs? As discussed in the introduction, the fields of maintenance aiding and training are closely related, so closely that the same investigators work in both fields and that they often mention the utility of an approach in one field to the other. The key relating these two fields is the *model-based approach* to diagnosis. Given this approach, a device model can be interchangeably employed as a job aid or a training aid without fear of falling prey to the "retrofitting" issue. MATIE demonstrated this commonality with its "Tutor/Diagnose" button that instantly changed the operating mode of MATIE from lob aid to training system.

There are some constraints on this convertibility, however. As MATIE illustrated, it is possible to build a trainer that usefully trains the top-level, control knowledge involved in diagnosis, but does not train the instantiation of signal values and locations from technical documentation. Thus, MATIE could help trainees know when it was good to check the SCU output, but could not help them learn just where to test or what to expect there. In the job aiding modality, then, MATIE could recommend testing the SCU output, but could not carry out the test, lacking the knowledge needed to instantiate the test's location and value.

To summarize, in training it is possible to abstract away some parts of the troubleshooting problem that would need to be available for job aiding. Thus it would not always be possible to go from a model-based trainer to a job aid. But given that a job aid was model-based, *it should be sufficient basis for a complete training system.* If the job aid is not model-based (as was the original MATIE), being based on the establish/refine approach, it would not be convertible to a trainer and would be subject to "retrofit."

V. FUTURE DIRECTIONS

Future Directions of MATIE

Development of MATIE to this point has focused on the construction of a working circuit model and the representation of troubleshooting expertise in the expert module. Future efforts will allow MATIE to evolve along several other lines that will enhance its effectiveness as an ITS. Besides further development within each of the modules, enhancement of the interrelations and communication between modules will constitute a large part of the remaining effort. Figure 12 represents a future architecture for MATIE that reflects enhancements to its major functions. Suggested improvements for each of the functional components are discussed below.

The Expert Module

Relatively few changes will be required by the expert module. However, it does need to be expanded to allow for replacement of components and cables as a legitimate test action. MATIE now accepts replacement only as a final step--the "Solve." But when test costs exceed the effort and expense of simply replacing the unit, it is preferable to take the latter action. The expert module will need the ability to recognize those situations in which replacement is called for in the troubleshooting process, and include replacement as a legitimate troubleshooting activity.

The Student Module

The current student module will require considerable expansion along several dimensions in order to accurately diagnose and model increasing student expertise. It first needs to recognize when the student's strategies undergo major revisions, and when increased proficiency (learning) has allowed the student to operate in different problem-solving modes. For instance, inclusion of test costs and component failure rates in the student's concept of the circuit is a shift to a more predictive mode. Second, it needs to recognize appropriate points in the learning process to intervene as a coach and interject advice or discuss "issues" that can be used by the student to further enhance performance. This includes assessment of how frequently and under what conditions students require coaching and other assistance, as well as when various troubleshooting concepts have been mastered (e.g., uses the replacement action appropriately as both a test action and a solve, incorporates test cost and failure rate data into the test point analysis phase, and understands how suspected components are eliminated through "clear-back"). If commonly occurring errors or "bugs" can be anticipated, these will also be incorporated into the issue recognition process. Third, MATIE's student modeling capabilities will need to be expanded to allow tracking of student progress both within and across sessions. This will lead to a more accurate representation of student expertise and conceptual difficulties, thus permitting better tutoring by virtue of its inputs to the instructional environment and instructor module.

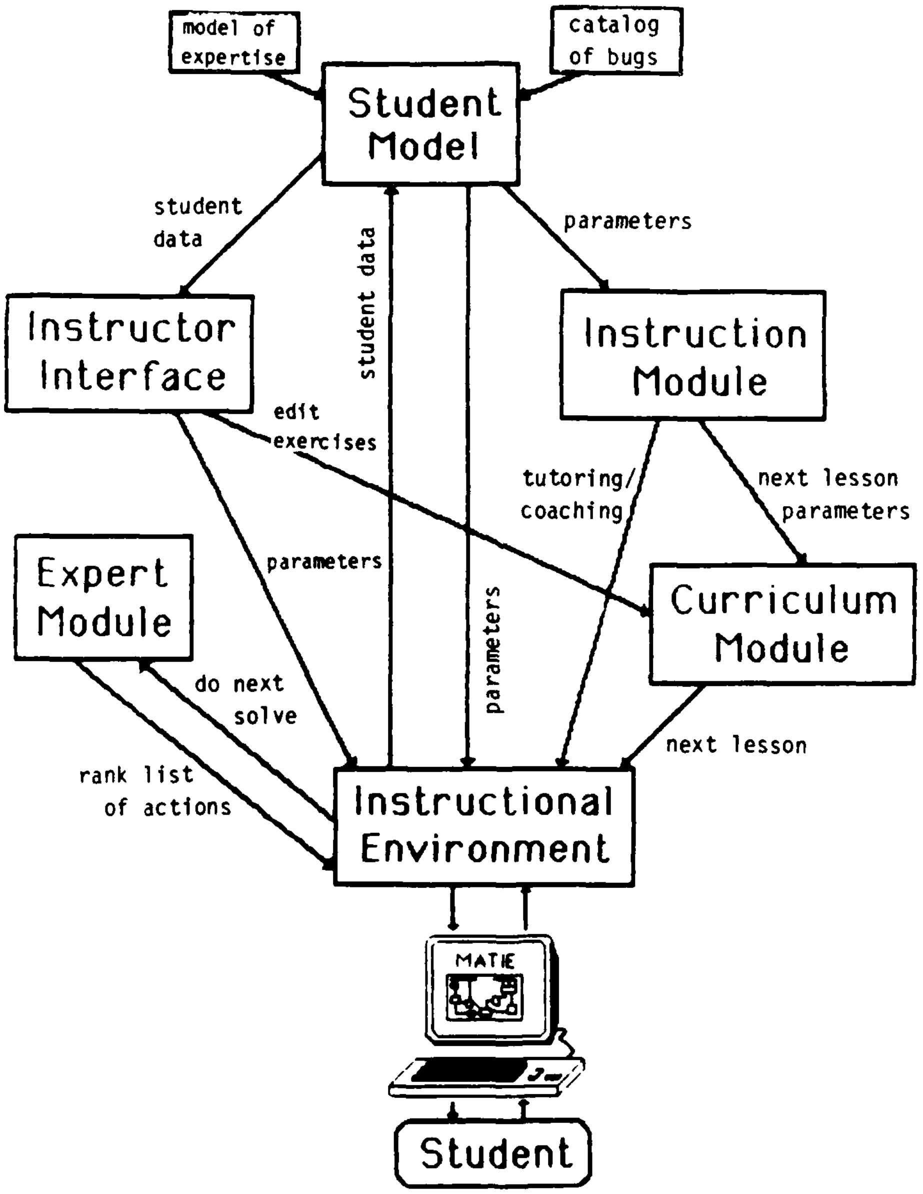


Figure 12. **The Future Architecture of MATIE.**

The Instructional Environment

To allow a more reactive learning environment, future versions of MATIE will need to dynamically adjust the parameters listed in the "Approach" section (switches for graphics and inclusion of test cost or failure rate data, and thresholds for advice and help). Because the instructional environment is the interface between other ITS modules and the student, it will have to be modified to access any new information in the redesigned modules. It will also serve a key role in sending data to the student model for subsequent diagnosis and representation.

The Curriculum Module

Now that a good platform has been developed for students to exercise their troubleshooting skills, more attention will be paid to curriculum issues to make sure that the transition between different levels of competence is handled smoothly. The curriculum module is principally concerned with pedagogic issues of the ITS (deciding what issues need to be taught, in what order they are to be presented, and when it is appropriate for the student to access them).

The current lesson data base is comprised of exercises, each of which has a different faulty TRU. By solving them through application of the diagnostic algorithm, students come to understand the more generic issues of computing entailments (knowing the implications of GOOD and BAD test results, evaluating potential refinements in selecting test points, and applying test cost and failure rate data to the troubleshooting process). They also learn specifics pertaining to the 6883's topologic features, and the application of sequential testing patterns.

Among the issues to be addressed in future versions of the curriculum module will be lesson sequencing strategies, inclusion of partially solved problems, and a breakdown of how to best introduce the test cost and TRU failure rate data. Each of these considerations will be a matter for empirical evaluation within the current 6883 test station domain. A further consideration will be the generalizability of the MATIE approach (*ATIE) for other test stations, other circuit simulations, and even for other knowledge domains. Finally, connections to the instruction module will need to be included in order for lesson selection to be programmed. Of the above considerations, two topics are worthy of further elaboration.

Integrating Time-Since-Replacement Data into Diagnostics. A theoretical diagnostic model which minimizes average costs of identifying faults is fairly well understood and has been incorporated in MATIE so far as practically possible given data which are available for all 6883 units. However, the optimal model, in calculating the probability of failure for each component, combines failure rates with time-since-replacement. For example, if a component has a mean time between failures (MTBF) of 1,000 hours and the component has been operational for 1,000 hours, it has a 0.5 probability of failure. Unfortunately, the probabilities at other values of time-since-replacement are not so straightforward, since the probability of failure is usually higher than expected for a new, untested component, and the probability distribution will differ for different kinds of components (e.g., cables versus CRTs). A completely satisfactory model would take each of these unique functions into account. Such completeness is unlikely to be achieved in most expert systems or expert modules of ITSs, and the complexity of such a model probably makes its teaching unrealistic as a goal for an ITS student. However, a substantial performance gain could be achieved by using a rough approximation, probably a linear function of time-since-replacement divided by MTBF, uniformly applied to all components.

Time-since-replacement is unique to each piece of equipment; so, it is much more difficult to use than is general information such as MTBF for components. The most valuable source of these data is in the equipment's service history, which should be increasingly available via electronic media. Even when the data are not available from records, experienced troubleshooters can often infer that a component has been replaced, from its physical appearance or signs of modification such as solder. More generally, they can estimate the unit's age or may know when production of the class of equipment ceased. They may also intuitively modify the "book" values for failure rates under many conditions; for example, when equipment has been used during sea operations which expose components to corrosion.

Memory and Computational Requirements of Diagnosis. Current models assume no penalty for the logical operations which must be performed to carry out various strategies, nor for the amount of memory required to keep track of past results and conclusions. For humans, both memory and computation pose significant limitations and relate closely to error rates and efficiency, particularly at the extreme level of complexity involved in model-based diagnostic reasoning using failure rates and test costs in every component. These considerations bear on both the choice of diagnostic strategies to train and the kinds of JPAs which should be provided.

The Instruction Module

The instruction module is responsible for communicating information to the student. This is presently accomplished through a coaching function which is activated according to a preset advice threshold set by the instructor. Future implementations of the instruction module, besides allowing this parameter to be dynamically altered based on input from the student model, should increase the degree of articulation offered as a means for improving diagnostic strategy. This can be supplemented by increased tutoring, in addition to its present role as coach. Thus, a more detailed rationale could be provided for test point selection, clear-back implications might be better described, and exchanges could be initiated to determine the source of student difficulties. Besides offering a greater degree of trainee assistance, this module should later govern the selection of lessons from the curriculum module.

The Instructor Interface

Future implementations of the instructor interface will need to provide an enhanced facility for course management activities. This will involve improvement of existing instructor interface capabilities, including access to the curriculum and instruction modules, and direct input to the instructional environment. Additions to the module will concern increased capabilities for handling larger numbers of students, a better means for dividing student data into multiple rosters, the use of statistics for later analysis of grouped data, and providing for the evaluation of ongoing performance.

Future Directions of Intelligent Tutoring Systems

Many current authors have suggested future directions for research and application in ITSs, including Richardson (1988), Anderson (1988), VanLehn (1988), Park et al. (1987), Halff (1988), and others. Although we will not attempt to list all the recommendations from these authors, we will summarize a few of the areas where future R&D is likely to produce high payoffs.

A direction that is strongly suggested by Halff (1988) and Park et al. (1987) is the integration of the progress in ITSs with the lessons learned over several decades of instructional development. As pointed out earlier in this report, the ITS represents a paradigm that has developed in considerable isolation from other disciplines involved in Instructional Systems Development (ISD), Computer-Based Training (CBT), programmed instruction, and educational technology. While tutoring presents special demands that have not been adequately addressed by these other disciplines, there is still a great deal that can and should be incorporated from these areas, specifically in the areas of development methodologies, structuring of curricula and instruction, measurement and validation, and principles of learning and behavior.

Development Methods

Well-defined guidance for developers of instructional and training material of various forms exists to differing degrees within the ISD process. ISD separately details five phases in the life cycle of an instructional development effort: analysis, design, development, implementation, and evaluation. Several subphases in each major phase are also detailed (see Table 1).

Table 1. **Phases of the ITS Process**

Phase I - Analysis	Define jobs and tasks, select tasks for training, construct job performance measures, identify previously developed training material and information in the task domain.
Phase II - Design	Develop detailed specification of the form and content of instruction and curriculum, assess entry behavior criteria.
Phase III - Development	Determine methods of instructional management, choose the student activities and learning experiences, determine the form and content of the delivery system, review and adapt existing material, systematically design instruction, test and evaluate performance of the instruction.
Phase IV - Implementation	Implement instructional management plan, conduct instruction.
Phase V - Control	Conduct internal evaluation, conduct external evaluation, revise system.

ISD was formalized in the military (AFM 50-2, 1975) before much work had been done with ITSs, but it incorporated much of the philosophy that had evolved up to that time within the instructional community. More variability has appeared in the development processes of the ITSs developed up to the present. Because computer-based coaches and tutors place different demands on instructional developers than do the more conventional frame-based CBT and interactive videodisc, the development methodologies would be expected to differ to some degree. Experimentation has been the rule with respect to development methods in the ITS arena. A style of development known as "exploratory programming" (Sheil, 1983) has been applied to ITS development (Pliske & Psotka, 1986), because the available hardware and software tools have been powerful enough to support it.

However, ITS developers need to recognize that there is a valuable source of lessons learned within the ISD methodology that should be applied wherever possible. Future ITS efforts should include among their goals the systematic development of methodological extensions to prior approaches such as ISD to take advantage of the combined experience of the ITS tradition as it accumulates.

Structuring Curriculum

As mentioned earlier, Halff (1988) has suggested that the structuring of curriculum and instruction is an area in which the literature of instructional technologies contains a rich heritage that should be incorporated in the methodology of ITS. The recommendations of theorists like Englemann and Carnine (1982), Gagne (1973), Gilbert (1962), Markle (1964), Tennyson, Woolley, and Merrill (1972), and others should be helpful in structuring the instructional content. The phase of development known as analysis in the ISD process roughly corresponds with knowledge engineering in ITS development, and should produce a knowledge structure that suggests sequences or dependencies based on prerequisite concepts or construction of response chains. In the case of teaching a concept, the concept's features can be analyzed to construct an optimal sequence of examples and non-examples that illustrate the concept.

Better Measurement and Validation

Instructional technology has evolved methods for evaluating instruction, as it is being developed and after it is fielded, to assure that it accomplishes its objectives. In the early era of ITSs, the objective was not always to teach effectively, but to demonstrate the potential of applying techniques from AI. In this case, it was sufficient to show that technology could be applied impressively. As the ITS technology matures, it will be called upon for more pragmatic results, and it will be essential to develop or adapt a methodology of validation so that weaknesses in design can be identified and corrected and so that claims of instructional effectiveness can be supported. Good methods of formative and summative evaluation (during and after development, respectively) have emerged in recent years and should be investigated as to their applicability to ITSs. Littman and Soloway (1988) have suggested that the student model in the ITS can be used in an evaluation methodology for ITSs; they differentiate between external evaluation (the effects of the ITS on the students) and internal evaluation (the effectiveness of the relationship between elements of the ITS architecture). More work is needed, since the suggestions are based on an incomplete theory of evaluation. These newer efforts to develop an evaluation methodology should incorporate elements from the evaluation phase of the ISD model, which is based on a solid foundation of educational theory and research.

Learning

ITS and AI researchers in general have been less concerned with the process by which knowledge is acquired than with the representation of the knowledge. As Halff (1988) and Park et al. (1987) have pointed out, ITS implementors could benefit greatly by taking learning explicitly into account in the design of their systems. This calls for more attention to existing literature within those branches of psychology where theories of learning and adaptive behavior have been and continue to be researched. Not only would this attention to learning benefit the student model and expert model, it would have considerable payoff in the sequencing of material at both the instructional and curricular levels.

A related need is to consider the relationship between two distinct types of learned behavior: that which is learned from verbal descriptions, definitions, and rules, and that which is learned by experience with the problem environment. ITS designers, like all other educators, have been faced with a basic choice between two strategies for teaching students: give them verbal/declarative knowledge or let them learn from experience and feedback. MATIE uses the latter strategy, giving the student direct experience with the desired behaviors, along with explanations. Recent research and theories indicate that there are significant implications of the choice between declarative and experiential strategies, which can also be labeled "rule-governed" and "contingency-shaped." As Hutchison and Stephens (1987) summarized, declarative teaching strategies are preferable when learning by experience is dangerous or difficult to arrange, when speed of learning is critical (experiential learning is generally slower), or when immediate feedback from experience is likely to keep students from learning from more important delayed feedback.

On the other hand, there are two major advantages favoring experiential training: higher level of competence and increased sensitivity to feedback. AI theorists (Winograd & Flores, 1987) and critics alike (Dreyfus & Dreyfus, 1987) have recently begun to recognize the seriousness of the long-recognized fact that for most knowledge domains, rules are only approximations of true expertise. Trainees learning by rules may achieve a minimal level of competence but not true expertise, which involves subtler relationships not expressible in rules. This weakness of rules might still not be too serious if graduates of training could be expected to learn the more subtle knowledge quickly from on-the-job experience. However, a substantial body of recent research (Hayes, in press; Hayes, Brownstein, Zettle, Rosenfarb, & Korn, 1986) is demonstrating that people who learn to perform by following rules become remarkably insensitive to the consequences of their actions. It is as though they learn that they need only to pay attention to the rule. Trainees who have learned from direct (or simulated) experience, on the other hand, quickly adjust their behavior in line with feedback from their actions. These two disadvantages of rules generally lead one to prefer ITS tutoring strategies which have a strong experiential component to them (such as MATIE), unless what is to be learned is, in fact, primarily verbal.

New Architectures in ITSs

Very recently a great deal of interest in the AI world has focused on the potential of neurocomputing or adaptive network systems (ANSs) as an alternative to symbolic approaches for many application domains. Their potential for application in ITSs is a very timely issue to address, particularly since members of the current project team are also active members of the neural networking community.

As noted above, there is a growing recognition that for many knowledge domains, rules are incapable of representing the complex and subtle relationships in expert performance. Neural networks, on the other hand, are designed to represent the "network of relationships" in a way very similar to how humans learn them. Thus, ANSs seem to have obvious potential for replacing symbolic systems both for the expert model and the student model (and possibly the tutor model).

A second relevant capability of ANSs is that they can learn directly from contact with the subject matter (generally a data base containing historical data from the domain), thus avoiding a large part of the painstaking knowledge engineering that is required for symbolic AI. For example, a MATIE knowledge base could be constructed by generating a set of fault conditions, each containing the measurements which would be found along with the fault responsible. For most domains, the exemplar set does not need to contain all possible combinations of conditions (which is often an intractable number and would reduce the solution to a table lookup), but enough to allow the system to generalize to unseen cases.

A third relevant feature of ANSs is that they inherently utilize benefit and cost/effort information, permitting them to optimize performance strategies. For example, diagnostic skills such as those in MATIE should take into consideration the cost of each possible test and the likelihood that each component will fail, along with the logical relationships used in fault isolation. Since most knowledge domains involve these considerations, ANS offers a potentially significant advantage.

On the negative side, the very complexity of ANS systems causes difficulties. For example, there is more information in an ANS representation than in a symbolic representation which approximates it. In defense of the ANS, the complexity is a result of more accurately modeling the domain; but it would be more difficult to teach and to explain from such a model if the same tutoring approaches were attempted as in some ITSs (ones using declarative teaching strategies, discussed above). On the other hand, if such direct teaching and explanation were not required (as in experiential teaching), many of these problems would disappear. The previous discussion of the other advantages of experiential teaching strategies suggests that there may be a good match between ANS techniques and experiential ITSs. Figure 13 illustrates the concept of adaptive network systems applied to diagnostics.

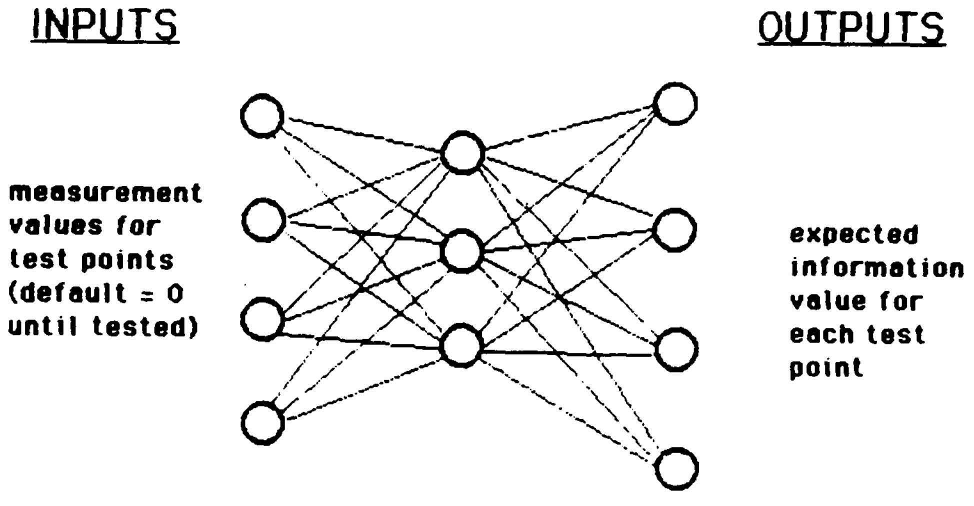

INPUTS
OUTPUTS
measurement values for test points (default = 0 until tested)
expected information value for each test point

Figure 13. **Diagnostic Expert System Using ANS.**

Future Directions of Maintenance ITSs

The need for generalized troubleshooting stems from the impossibility of providing system-specific training for all eventualities (due to the increasing complexity of systems). Generalized troubleshooting skills are often gained through extensive field experience, but not dependably so. Thus, there is a need to consciously provide focused training to promote the development of generalized troubleshooting skill. Short of one-on-one tutorial interaction, there is little alternative to intelligent maintenance training systems for the provision of the type of experiences needed to develop generalized troubleshooting expertise.

The experiences reported herein reinforce the view that the model-based expert system should serve as the expert module of an intelligent maintenance training system. It was seen that the model-based approach is the fundamental reasoning method for device diagnosis and that this approach is inherently device-independent. This finding not only reconfirms the validity of a generalized troubleshooting training objective, but it also provides an approach, as exemplified by MATIE and other model-based intelligent maintenance training systems.

A major conclusion of this effort concerns the convertibility of ITSs into job performance aids and vice versa. This convertibility is preconditioned on the use of a model-based approach to diagnosis. Without it, such interconversion is not possible. It was also noted in the conclusions that a job aid requires the signal location and value instantiation capability that may be omitted, as was the case for MATIE, from a training system. When this capability is omitted, it is not possible to convert the training system to a job aid without augmenting the system with the signal instantiation mechanism.

Significant efficiency gains associated with training and aiding follow from the finding that model-based job aids can be converted into training systems and vice versa. The costs involved in the development of the device model are shared between these two applications. Moreover, this effort has explicated the theoretical basis and provided some empirical evidence to support the expectation of obtaining this efficiency gain.

Future Directions of Maintenance Training

It was shown in the introduction that maintenance training, as one of the major diagnostic resources, will remain an important part of weapon system logistical support. This finding is based on the tenets of the integrated diagnostics philosophy. All diagnostic resources (design for testability, built-in test, automatic test equipment, trained personnel, and technical documentation) play a role in supporting maintainability, and all of these resources must be combined in the appropriate mix depending on the circumstances pertaining to a given weapon system.

It should be noted that the trained technician is the diagnostic resource of last resort. Should all other systems fail, the technician is relied on to return the system to operational status. At times, all other systems will fail, because each is an engineering solution, based on engineering analyses; and it is inevitable that the abstractions and models used by the engineers will omit some important aspect of system functionality that will play a role in generating an unexpected failure mode. When the unexpected occurs, the maintenance technician fills the breach.

Several important conclusions can be drawn with respect to the training and aiding functions in the Air Force from this effort. First, given the close, almost reciprocal relationship between model-based trainers and aids, it should be concluded that an opportunity exists to better integrate Air Force training and aiding functions. That is, this methodology provides an excellent opportunity to provide detailed, quality, interactive, job-relevant, on-the-job training. Second, given the discussion in the introduction and background about training requirements, it is concluded that classroom training for troubleshooting should focus on developing generalized troubleshooting skills. It is recognized that these must be developed in physical contexts, as discovered by Rouse (1984). But by using MATIE and covering troubleshooting in several technologies (hydraulic, etc.), it should be possible to meet an instructional objective of generalized troubleshooting ability. Corollary to this conclusion is the recommendation that device-specific training be provided on the job, through a training modality of the on-the-job maintenance aids.

It is believed that the conclusions and recommendations above have strong theoretical justification. It has been seen that the overarching philosophy of integrated diagnostics, an Air Force policy, reinforces the credibility of these results. This is sufficient basis for moving on with a program to develop and institutionalize this approach to training and aiding. But such a program must be guided by empirical evaluation of the claims made by this and other analyses. Not only will the program be reassured of its sound basis, but data germane to specific design alternatives and implementations will be available to guide the program. Accordingly, it is recommended that the MATIE effort or efforts like it be continued, providing for further elaboration of the modules of the ITS and the formative evaluation of the overall system in classroom and on-the-job training settings.

Future Directions of Maintenance

What are the future directions for maintenance as an element of logistical support? Future systems will emphasize "soft" failures over "hard" ones, where "graceful degradation" and fault tolerance are supported by redundant and reconfigurable systems built of small, standardized, line replaceable units (LRUs). Many of these features will appear in the Advanced Tactical Fighter. Will these features obviate the need for maintenance?

Actually, these features institutionalize maintenance doctrine in systems design. Maintenance is becoming a part of the system, not a part of its support. From design for testability and maintainability, to built-in test, reconfigurable LRUs, portable maintenance aids, and advanced maintenance information and training systems, the overall activity involved with maintenance is becoming progressively integrated with system design. As first pointed out by Genesereth (1984), the device model (the basis of MATIE) serves as a common data base, an interface standard for coordinating all the different perspectives of system design, deployment, and support. Design for testability is best approached using the model-based approach. Built-in test can be very effectively based on model-based diagnosis, especially system-level, top-down test, a crucial element in avoiding unnecessarily large false alarm rates from bottom-up built-in test. Finally, maintenance aiding and training are naturally based on the device model.

This overall movement is predicated on the need to maximize availability, by maximizing reliability and, when necessary, maximizing maintainability. Marginal improvements are greatest early in the system life cycle Thus, major efforts on design for testability and maintainability will pay the biggest dividends. Efforts on built-in test and recovery capability yield the next largest bonus, and so on. But as stated repeatedly, the overall solution to maintenance will always involve maintenance personnel. It is important to provide the best training and job aiding possible. It is the maintenance personnel that begin the design feedback loop from the field. The field impacts of poor design, built-in test, etc. are identified and dealt with by the field technician. Technicians in the field must be supported with a rich maintenance information system, such as IMIS (Link, Von Holle, & Mason, 1987), that provides technical and operational data and problem recording and management so that each technician may profit from shared experience, and so that the specific instances of fault isolation activity and the knowledge gained by one may be shared by all. These experiences will probably be stored in the information system as symptom-based heuristics to augment the model-based approach of a hybrid job aiding methodology. They will also serve as the feedback to system designers concerning additions or changes necessary in the device model to avoid repeated occurrence of those malfunctions which fall to the maintenance technician as diagnostician of last resort.

In sum, the future of maintenance portends increasing reliance on the device-model-based and model-based reasoning for all phases of weapon system procurement and support. A move by the training community toward model-based training methods is not an independent move and is not a move away from the overall trend in system design. Indeed, such a move is and shall be intrinsically coordinated with the technological advances being made in system design.

REFERENCES

Anderson, J. (1988). The expert module. In M. C. Polson & J. J. Richardson (Eds.), *Foundations of Intelligent Tutoring Systems* (pp. 21-53). Hillsdale, NJ: Lawrence Erlbaum Associates.

Atkinson, R.C. (1972). Ingredients for a theory of instruction. *American Psychologist, 27*, 921-931.

Atkinson, R.C. (1976). Adaptive instructional systems: Some attempts to optimize the learning process. In D. Klahr (Ed.), *Cognition and instruction*. New York: Wiley.

Bobrow, D.B. (1984). *Qualitative reasoning about physical systems*. Cambridge, MA: The MIT Press.

Bonar, J., Cunningham, R., & Schultz, J. (1986, September). An object-oriented architecture for intelligent tutoring systems. *Proceedings of the First Annual Conference on Object Oriented Programming, Systems, Language and Applications* (pp. 269-276). Portland, OR.

Brown, J.S., Burton, R.R., & Bell, A.G. (1975). SOPHIE. A step towards a reactive learning environment. *International Journal of Man-Machine Studies, 7*, 765-696.

Brown, J.S., Burton, R., & de Kleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. In D. Sleeman & J. S. Brown (Eds.), *Intelligent Tutoring Systems*. New York: Academic Press.

Burton, R.R., & Brown, J.S. (1982). An investigation of computer coaching for informal learning activities. In D. Sleeman & J. S. Brown (Eds.), *Intelligent Tutoring Systems*. New York: Academic Press.

Carbonell, J.R. (1970). AI in CAI: An artificial intelligence approach to computer assisted instruction. *IEEE Transactions on man-Machine Studies, 11*, 190-202.

Chandrasekaran, B. (1983). Towards a taxonomy of problem solving types. *The AI Magazine, 4*, 9-17.

Cicchinelli, L.F., Harmon, K.R., Keller, R.A., & Kottenstette, J.P. (1980). *Relative cost and training effectiveness of the 6883 three-dimensional simulator and actual equipment* (AFHRL-TR-80-24, AD-A091 808). Lowry AFB, CO: Logistics and TechnicalTraining Division, Air Force Human Resources Laboratory.

Cicchinelli, L.F., Harmon, K.R., & Keller, R.A. (1982). *Relative cost and training effectiveness of the 6883 F-111 converter/flight control system simulators as compared to actual equipment* (AFHRL-TR-82-30, AD-A123 534). Lowry AFB, CO: Logistics and Technical Training Division, Air Force Human Resources Laboratory.

Clancey, W.J. (1986). From GUIDON to NEOMYCIN and HERACLES in twenty short lessons. *AI Magazine, 7*, 40-60.

Clancey, W.J. (1987). *Knowledge-based tutoring*. Cambridge, MA: MIT Press.

Clancey, W.J., & Letsinger, R. (1981). NEOMYCIN: Reconfiguring a rule-based expert system for application to teaching. In Proceedings of the 7th International Conference on Artificial Intelligence. Vancouver, BC.

Crowder, N.W. (1959). Automatic tutoring by means of intrinsic programming. In E. H. Galanter (Ed.), *Automatic teaching: The state of art*. New York: Wiley.

Department of the Air Force. (1975). *Instructional system development*, Air Force Manual AFM 50-2. Washington, DC.

Dreyfus, H.L., & Dreyfus, S.E. (1987). *Mind over machine*. New York: The Free Press.

Englemann, S., & Carnine, D. (1982). *Theory of instruction: Principles and applications*. New York: Irvington.

Fink, P.K., Lusth, J.C., & Duran, J.W. (1984). A general expert system design for diagnostic problem solving. In *Proceedings of the IEEE Workshop on Principles of Knowledge-Based Systems*. Institute of Electrical and Electronics Engineers, Denver, CO, IEEE Computer Society Press.

Freedman, R.S., & Rosenking, J.P. (1986). Designing Computer-based training systems: OBIE-1:KNOBE. *IEEE Expert, 1*, 31-38.

Gagne, R.M. (1973). Learning and instructional sequence. In F.N. Kerlinger (Eds.), *Review of research in education*. Itasca, IL: F. E. Peacock, Inc.

Gagne, R.M. (1977). Analysis of objectives. In L.J. Briggs (Ed.), *Instructional design: Principles and applications*. Englewood Cliffs, NJ: Educational Technology Publishers.

Genesereth, M.R. (1984). The use of design descriptions in automated diagnosis. *Artificial Intelligence, 24*(1-3), 411-436.

Gilbert, T. F. (1962). Mathetics: The technology of education. *Journal of Mathetics, 1*, 7-73.

Halff H.M. (1988). Curriculum and instruction in automated tutors. In M.C. Polson & J.J. Richardson (Eds.), *Foundations of intelligent tutoring systems* (pp. 79-108). Hillsdale, NJ: Lawrence Erlbaum Associates.

Hayes, S.C. (in press). *Rule-governed behavior: Cognition, contingencies, and instructional control*. New York: Plenum Press.

Hayes, S.C., Brownstein, A.J., Zettle, R.D., Rosenfarb, I., & Korn, Z. (1986). Rule-governed behavior and sensitivity to changing consequences of responding. *Journal of the Experimental Analysis of Behavior, 45*, 237-256.

Hutchison, W.R., & Stephens, K.R. (1987). Integration of distributed and symbolic knowledge representations. *Proceedings of the First International Conference on Neural Networks, 3*, 395-398. San Diego, CA.

Jones, K.R., Bamford, D.E., Richardson, J.J., & Sizer, T.M. (1987). *Maintainer's associate training instructional environment, final report: Phase I*. Brooks Air Force Base, TX: Air Force Human Resources Laboratory.

Kearsley, G.P. (1987). *Artificial intelligence and instruction*. Reading, MA: Addison-Wesley.

Keller, R.A. (1985). *Human troubleshooting in electronics: Implications for intelligent maintenance aids* (AFHRL-TP-85-34, AD-A161 832). Lowry Air Force Base, CO: Training Systems Division, Air Force Human Resources Laboratory.

Kurland, L.C., & Tenney, Y.J. (1988). Issues in developing an intelligent tutor for a real-world domain: Training radar mechanics. In J. Psotka, L.D. Massey, & S.A. Mutter (Eds.), *Intelligent tutoring systems: Lessons learned*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Lahore, H. (1984). *Artificial intelligence applications to testability* (RADC-TR-84-203). Griffiss AFB, NY: Rome Air Development Center.

Link, W.R., Von Holle, J.C., Mason. D. (1987). *Integrated maintenance information system (IMIS): A maintenance information delivery concept* (AFHRL-TP-87-27, AD-A189 335). Wright-Patterson AFB, OH: Logistics and Human Factors Division, Air Force Human Resources Laboratory.

Littman, D.C., & Soloway, E.M. (1988). Evaluating ITSs: The cognitive science perspective. In M.C. Polson & J.J. Richardson (Eds.), *Foundations of intelligent tutoring systems* (pp. 209-242). Hillsdale, NJ: Lawrence Erlbaum Associates.

Markle, S.M. (1964). *Good frames and bad.* New York: John Wiley and Sons.

Merrill, M.D., & Tennyson, RD. (1977). Teaching concepts: An instructional design guide. Englewood Cliffs, NJ: Educational Technology Publishers.

National Security Industrial Association. (1986). *Proceedings of a Conference on Implementation of Integrated Diagnostics.* Washington, DC: National Security Industrial Association.

Orlansky J., & String, J. (1983). Cost effectiveness of maintenance simulators for military training. *Machine Mediated Learning, 1,* 41-63.

Park, O., Perez, R.S., & Seidel, R.J. (1987). Intelligent CAI: Old wine in new bottles, or a new vintage? In G. P. Kearsley (Ed.), *Artificial intelligence and instruction* (pp. 11-45). Reading, MA: Addison-Wesley.

Pieper, W.J., Richardson, J.J., Harmon, K.R., Keller R.A., & Massey, R.H. (1984). *Interactive graphics simulator: Design, development, and effectiveness/cost evaluation* (AFHRL-TR-84-38, AD-A149 417). Lowry AFB, CO: Training Systems Division, Air Force Human Resources Laboratory.

Pliske, D.B., & Psotka, F. (1986) Exploratory programming environments for designing ICAI. *Journal of Computer-Based Instruction, 13,* 52-57.

Polson, M.C., & Richardson, J.J. (1988). *Foundations of intelligent tutoring systems.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Psotka, J., Massey, L.D., & Mutter, S.A. (1988). *Intelligent tutoring systems: Lessons learned*. Hillsdale, NJ: Lawrence Erlbaum Associates.

Richardson, J.J. (1983). *Artificial intelligence: An analysis of potential applications to training, performance measurement, and job performance aiding* (AFHRL-TP-83-28, AD-A133 592). Lowry AFB, CO: Training Systems Division, Air Force Human Resources Laboratory.

Richardson, J.J. (1985). *Artificial intelligence in maintenance: Proceedings of the Joint Services Workshop.* Park Ridge, NJ: Noyes Publications.

Richardson, J.J. (1987). *Proceedings of the Air Force workshop on artificial intelligence applications for integrated diagnostics* (AFWAL-87-1094). Wright-Patterson AFB, OH: Air Force Wright Aeronautical Laboratories.

Richardson, J.J., Anselme, C.J., Harmon, K.R., Keller, R.A., & Moul, B.L. (1986). *Artificial intelligence technology for the maintainer's associate* (AFHRL-TR-86-31, AD-A715 211). Brooks AFB, TX: Training Systems Division, Air Force Human Resources Laboratory.

Richardson, J. (1988). Directions for research and applications. In M. C. Polson & J.J. Richardson (Eds.), *Foundations of intelligent tutoring systems* (pp. 243-252). Hillsdale, NJ: Lawrence Erlbaum Associates.

Richardson, J.J., Keller, R.A., Maxion, R.A., Polson, P.G., & DeJong, K.A. (1985). *Artificial intelligence in maintenance: Sythesis of technical issues* (AFHRL-TR-85-7, AD-A160 863). Lowry AFB, CO: Training Systems Division, Air Force Human Resources Laboratory.

Rouse, W.B. (1985). Models of natural intelligence in fault diagnosis tasks: Implications of training and aiding of maintenance personnel. In Richardson (Ed.), *Artificial Intelligence in Maintenance: Proceedings of the Joint Services Workshop*. Park Ridge, NJ: Noyes Publications.

Sheil, B. (1983) Power tools for programmers. *Datamation*, 131-144.

Skinner, B.F. (1968). *The technology of teaching*. Englewood Cliffs, NJ: Prentice-Hall.

Sleeman D., & Brown, J.S. (1982). *Intelligent tutoring systems*. New York: Academic Press.

Suppes, P., Fletcher, J.D., & Zanotti, M. (1976). Models of individual trajectories in computer-assisted instruction for deaf student. *Journal of Educational Psychology, 68*, 117-127.

Tanner, M.C., & Bylander, T. (1984). Application of the CSRL language to the design of expert diagnostic systems: The Auto-Mech experience. In Richardson, J. (Ed.), *Artificial Intelligence in Maintenance:* Proceedings of the Joint Services Workshop. Park Ridge, NJ: Noyes Publications.

Tennyson, R.D., Woolley, F.R., & Merrill, M.D. (1972). Exemplar and nonexemplar variables which produce correct classification behavior and specified classification errors. *Journal of Educational Psychology, 63*, 144-152.

Towne, D. (1986). A generic expert diagnostician. *Proceedings of the Air Force Workshop on Artificial Intelligence Applications for Integrated Diagnostics* (pp. 218-237). Boulder, CO: University of Colorado, Center for Applied Artificial Intelligence.

VanLehn, K. (1988). Student Modeling. In M.C. Polson & J.J. Richardson (Eds.), *Foundations of intelligent tutoring systems*, (pp. 55-78). Hillsdale, NJ: Lawrence Erlbaum Associates.

Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Los Altos, CA: Morgan Kaufman.

White, B.Y., & Frederiksen, J.R. (1986a). Intelligent tutoring systems based upon qualitative model evolutions. *Proceedings of Fifth National Conference on Artificial Intelligence* (AAAI-86 *1*, 313-319). Philadelphia, PA.

White, B.Y., & Frederiksen, J.R. (1986b). *Progressions of qualitative models as a foundation for intelligent learning environments* (Report No. 6277). Cambridge, MA: BBN Laboratories, Inc.

Winograd, T., & Flores, F. (1987). *Understanding computers and cognition: A new foundation for design*. Norwood, NJ: Ablex Publishing.

☆ U.S. GOVERNMENT PRINTING OFFICE: 1989—661-041/00034