FUNCTIONAL DESCRIPTION OF A COMMAND AND CONTROL LANGUAGE TUTOR

David R. Eike and Thomas L. Seamster
Carlow Associates Incorporated
8315 Lee Highway
Fairfax, Virginia 22031

Walter Truszkowski
Code 522.3
Goddard Space Flight Center
Greenbelt, Maryland 20771

ABSTRACT

This paper describes the status of an ongoing project to explore the application of Intelligent Tutoring System (ITS) technology to NASA command and control languages. The primary objective of the current phase of the project is to develop a user interface for an ITS to assist NASA control center personnel in learning Systems Test and Operations Language (STOL). Although this ITS will be developed for Gamma Ray Observatory operators, it will be designed with sufficient flexibility so that its modules may serve as an ITS for other control languages such as the User Interface Language (UIL). The focus of this phase is to develop at least one other form of STOL representation to complement the operational STOL interface. Such an alternative representation would be adaptively employed during the tutoring session to facilitate the learning process. This is a key feature of this ITS which distinguishes it from a simulator that is only capable of representing the operational environment.

INTRODUCTION

This paper describes the status of an ongoing project to explore the application of Intelligent Tutoring System (ITS) technology to NASA control centers. The primary objective of the current phase of the project is to develop a user interface for an ITS to assist NASA control center personnel in learning Systems Test and Operations Language (STOL) with the aim of designing the ITS with sufficient flexibility so that its modules may serve as an ITS for other control languages such as the User Interface Language (UIL).

The paper first addresses nine broad areas of functionality that combine to produce an ITS. This presentation emphasizes that these functions may have different levels of implementation, from a very simple level to a complex one requiring considerable computational resources. This approach decomposes the ITS into functions that do not match the traditional ITS modules (see Figure 1 for the relationship between the functions and the ITS modules). The reason for this decomposition is to take a fresh look at ITSs from the perspective of NASA command language training needs. The nine functions are as follows:

1) *Initiating* the tutoring session
2) *Assessing* the student's status
3) *Presenting* the problem
4) *Monitoring* the student's performance
5) *Assessing* the student's goal
6) *Identify* the information to be tutored
7) *Adapting* tutor mode to student
8) *Tutoring* the student
9) *Updating* the student model

The last part of the paper presents the critical issues affecting the current phase. These issues include which modules will be developed first, which functions will be given the highest priority in the ITS, and the process for deciding on whether to use an intermediate form of representation for the tutoring of the control language and/or the objects being controlled.

INITIATING THE TUTORING SESSION

An ITS may initiate a tutoring session in response to one or more of the following events: 1) a request from the student for instruction; 2) a pre-defined schedule of tutoring; or, 3) detection by the ITS of a flaw in the

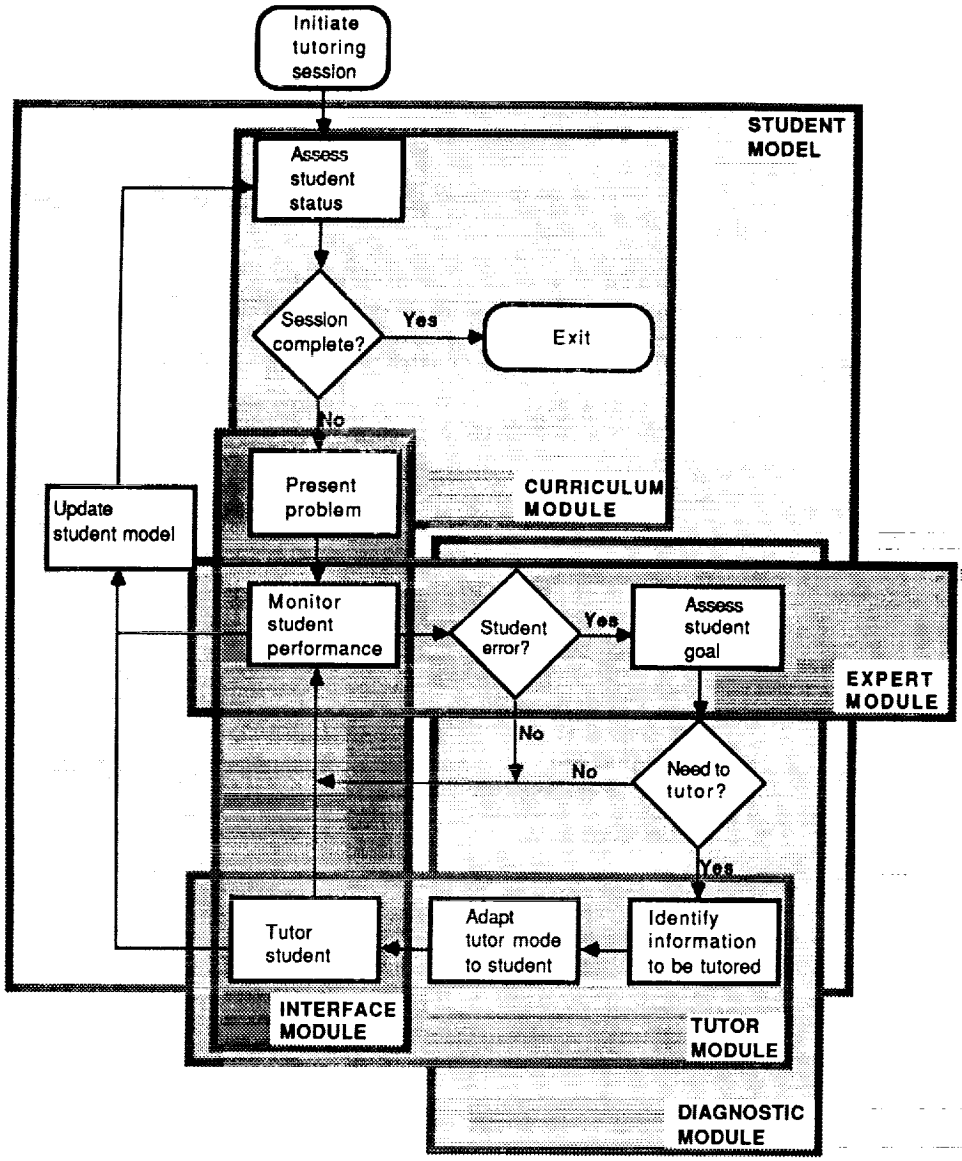

Figure 1. Functional Flow of a Command Language ITS

student's knowledge. Each of these approaches has significant implications for the design and overall functional characteristics of the ITS.

The simplest ITS architecture employs the programming equivalent of an "on" switch to initiate a tutoring session. In this approach, the student recognizes the need for tutoring and requests that the ITS begin tutoring. At a more complex level, the ITS may have the capability to infer what instruction is necessary by examining the student model.

An increasingly complex ITS would include the use of a tutoring schedule which describes the timing and sequencing of individual "lessons." The schedule is developed as part of the ITS's curriculum module, and modified, as necessary, to accommodate requirements of the individual student. The ITS queries the student model to establish the student's position within the schedule, and then initiates the appropriate tutoring session to advance the student to the next scheduled level of learning.

ASSESSING STUDENT'S STATUS

The ITS must infer the student's status within the session as well as the entire tutorial. Depending on the modules of an ITS, this assessment may be based on the interaction of the data in the student model with that of the curriculum module. At the start of a session, this assessment will be instrumental in determining which problem to present. This assessment serves also to determine if the session is complete.

PRESENTING THE PROBLEM

In its simplest form, this function involves selecting the next problem from a predetermined problem set. The assumption here is that the ITS's primary function is to present domain related problems to the student, and to provide tutoring when the student has difficulty with the problem. A more sophisticated implementation of this function would take into account a number of factors in the student model in order to more closely tailor the problem selection and representation process to the individual student. Such elements as previous errors, difficulties with specific concepts, and the recency with which the student has used the tutor could combine to guide the optimum problem selection.

MONITORING THE STUDENT'S PERFORMANCE

This function requires that there be a common workspace for the ITS and student, and that the ITS is capable of monitoring the student's performance in some meaningful context. At least five levels of student monitoring are possible: keystroke, word, phrase, consequence, and complete student product.

There are obvious advantages and disadvantages for each of the above levels. Burns and Capps (1988) refer to this as the "bandwidth question." At the level of individual keystrokes, the ITS may achieve the greatest precision. The cost of this precision is registered in terms of development effort and potential for "abuse." In terms of development effort, the cost is associated with developing models of the user and the system which can be meaningfully examined at the level of individual keystrokes. In terms of "abuse," the keystroke level of monitoring provides the greatest opportunity for intrusive intervention, what might be termed the "backseat driver" effect. As the level of monitoring increases in granularity, the advantages and disadvantages shift accordingly, until at the total product level, the ITS may be too imprecise to be effective, but is virtually incapable of intrusive intervention.

ASSESSING STUDENT'S GOAL

If the student's performance deviates from that of the expert module, or is in error for some other reason, the ITS must assess the student's goal for the current transaction. In the present context, this will be necessary to evaluate the adequacy of the student's semantic knowledge of STOL. At a minimum, the ITS should be able to infer the student's high-level goal based on the current problem.

A more accurate definition of the student's goal may be achieved by incorporating data on the student's mental models of the current problem domain. The ITS may generate an hypothesis about how the student conceptualizes the current problem, both as an individual entity and as part of the problem domain. This hypothesis should describe both the student's problem conceptualizations as well as problem solving strategies.

The student model traditionally supports this function by providing two types of information. First, the student model provides a general description of how the typical student conceives of and solves specific problems in the problem domain. Second, the student model contains an historical record of the current student, including aptitudes, the content and outcome of previous tutor sessions, typical errors, and preferences for tutor session design and content. This information in used to infer the student's current goal and to formulate preliminary hypotheses about the student's current level of knowledge.

Given a student goal, the ITS can determine if tutoring is required. Not every student error will require tutoring. For example, an ITS may be capable of distinguishing between errors and what Carroll and McKendree (1987) refer to as "characteristic nonoptimalities" (i.e., idiosyncratic problem solving strategies which are effective but not optimal). For this capability, the ITS must be sensitive to individual problem solving strategies which may range outside the normative model, but which are effective and consistent with both the both student's objectives and his conceptualization of the problem. The concern here is not only with regard to the student's perception of the acceptability of the ITS, but also in terms of maximizing the student's natural propensity to learn. Woolf and Cunningham (1987) recommend developing an ITS which can guide a students without interruption as long as the student appears to be attaining a specific goal.

SELECTING THE INFORMATION TO BE TUTORED

At a minimum, the ITS selects information to be imparted to the student based on explicit errors which are indicative of a specific flaw in the student's knowledge. At a practical level, the ITS can infer knowledge gaps by comparing the student's performance to that of the expert using a normative model which reflects changes in student behavior as they acquire skills and knowledge. In general, this approach assumes that the knowledge gaps of the student can be inferred once the student's position on a hypothetical skill acquisition continuum has been defined. This approach is implemented through the development of a general model of student performance which describes changes in the way the student solves problems at various stages of expertise. Given such a model, we can elaborate on individual stages using system specific examples. A major limitation of this approach is that it must assume that the knowledge level of the student is monolithic for a particular stage of learning. This limitation may be at least partially overcome by including independent student models for each relevant knowledge domain. This solution, however, raises significant implementation issues regarding efficiency and economy.

Wenger (1987) proposes a different solution to this problem which involves development of a diagnostic model of the student's current skills and subskills. This model is based on a representational scheme for procedural skills with an emphasis on those skills that can be mislearned. The notion of evaluating individual student actions, in terms of the implications of those actions for the skill/knowledge level of the student, may be thought of as an assessment of the ignorance of the student. The term, ignorance, is used here in its clinical sense without pejorative connotation. What is required, then, is a descriptive and computational model which can be used to evaluate the type and amount of ignorance an individual possesses with regard to a particular element of the knowledge domain. This shifts the focus of the ITS's assessment of the student from what the student knows to what the student does not know. This assessment may more directly support the selection of knowledge to be tutored than does the development of traditional student models. This requirements for assessing ignorance are discussed in detail in Eike and Seamster (1989).

There are at least 4 distinct states of ignorance which must be considered:

1. Unknown/misknown class of elements - the student is not aware of the existence of an entire class of elements or has mistaken notions about the class. In this case, the ITS will have to explain to the student what are the characteristics which define membership in this class.
2. Unknown/misknown element - the student is not aware of the existence of the element or does not believe that the element is a member of a particular class. The student has a valid schema for the class of elements to which the current item belongs, but is unaware of the existence of this particular instance of that class or has not assigned the element to the proper class.
3. Unknown/misknown relevance - the student is aware of the existence of the object but has failed to observe the relevance of the object for the present problem. The basis for this ignorance may be more profound, necessarily deriving from a failure to recognize the relationship between known attributes of known objects.
4. Unknown/misknown rules or procedures - the student is aware of all of the relevant objects and classes of objects, but lacks or misunderstands the rules necessary to solve the problem.

In order to instantiate this approach, the ITS needs the capability to detect an error and then infer the underlying ignorance. In selecting the information to be imparted, the ITS may consider several information parameters, including content, amount, level of detail, and format. As shown in Figure 1, the diagnostic module interacts with the student model and the curriculum module in making this selection.

ADAPTING TUTORING MODE TO STUDENT

A major distinction between conventional Computer Assisted Instruction (CAI) and an ITS is the ability to dynamically adapt the tutoring strategy to the current needs of the student. In theory, one of the most powerful features of an ITS is the ability to alter its mode of instruction to accommodate the unique requirements of the individual student. In practice, however, this capability may be beyond the current state of the art. Carroll & McKendree (1987) observe that current ITSs are not able to reason about tutoring or select strategies dynamically.

Wenger (1987) describes a number of potential tutoring strategies, including the following:

Case method
Coaching
Engage and pull
Issues and examples
Model tracing
Modeling-scaffolding-fading
Planning nets
Steering testing
Socratic method

The current concern is in developing a partially adaptable ITS. In the case of a STOL ITS, the main issue it whether the ITS should have more than one tutoring strategy, and how would several strategies be selected and presented to the student. Norcio and Stanley (1988) discuss several negative aspects of adaptive interfaces which have implications for ITS design. First, adaptation, by definition, involves a change in the way the ITS interacts with the student. According to Norcio and Stanley, such changes may inhibit the user's ability to develop a coherent model of the system. This has the potential effect of undermining the student's confidence in his understanding of the system with a consequent degrading of the student's performance. Similarly, the student may experience a sense of losing control over the system such as not being able to predict the system's response. This also may contribute to a general feeling of confusion on the part of the student. From a practical perspective, adaptation imposes significant development costs.

TUTORING THE STUDENT

Given that the ITS has identified and diagnosed the student's problem, and developed a general plan for remedial action, the system is ready to begin advising or tutoring the student. This function contains those elements of the ITS that involve the actual communication of information to the student. Irrespective of the tutoring strategy employed by the ITS, the system should have the ability to explain the rules and information processes which underlay its knowledge base. This feature is typically referred to as the "glass box model" (Burns and Capps, 1988), and is similar to "explanation facility" used in expert systems.

In order to be maximally useful to the student, the ITS should have the following capabilities relative to explaining the rules contained in its knowledge base:

1) At any point during a tutoring session, the ITS should be capable of displaying the rules which are central to solving the current problem.
2) The ITS should be capable of recalling and displaying each invoked rule and associating it with a specific event to explain the rationale for the ITS's assessment of the event.
3) The tutor module should be able to search the knowledge base to locate rules or items of knowledge in response to specific inquiries from the student.

The ITS should provide the capability to model and predict the performance of an expert in solving a particular problem. In this manner, the student could observe the expert's problem solving strategies in a context that is relevant to the student. The tutor module would then be able to "play back" the expert's solution, step by step, with the student examining each step and querying the ITS for explanations and justifications.

As indicated in Figure 1, the tutoring function involves the tutor module, the student model, and the user interface module.

UPDATING THE STUDENT MODEL

The final function of the ITS is to update the student model. The student model should be updated based on the results of the activities which occurred during the session. Figure 1, depicts the evaluation function as being driven by the output of the tutor module in conjunction with the data on the student's performance. This is a somewhat simplified version of the update function. For a more complex version, information collected or generated by most of the remaining modules throughout the session could be incorporated into the update. This more complex form of updating could pose difficult data management problems.

CRITICAL ISSUES

During this phase of ITS design, there are a number of critical decision that are being made affecting the direction and final capability of the STOL ITS. A major decision is to emphasize the user interface. A number of the earlier ITSs initiated with the development of an expert module. This is due in part to the fact that some of these ITSs evolved from expert systems and consequently had established expert modules. In the case of STOL, there is not an existing expert system, and it was decided to start off with the development of whichever module would provide the best data for evaluating the feasibility of the project.

ITS development has matured to the point that greater emphasis can now be placed on the end-user as well as the ITS's impact on the training system. Given this new focus on the end-user, it was decided that the user interface module would provide a good starting point. A user interface prototype could be used to gather the traditional interface preference and performance data, and could be used to evaluate the relative effectiveness of alternative tutoring strategies in the context of specific student problems.

In discussing the development process of existing ITSs, it was discovered that some of the current ITSs were developed starting with the interface. Specifically, the Geometry Tutor (Anderson, Boyle, & Yost, 1985) was developed with the interface being completed first. One of the reasons for this is that the user interface for the Geometry Tutor is relatively complex, and the representation of the geometry proofs was considered to be a critical factor in the success of the Geometry Tutor. Analysis of this development approach revealed an additional benefit from starting with the user interface. If the user interface prototype were sufficiently flexible and robust, that prototype could be used to not only gather user data, but could be used as the primary tool for knowledge acquisition. STOL experts would interact with several sample problems and the various forms of representing STOL, and would provide a range of problem solutions as well as ways of optimizing STOL representations.

This early emphasis on the user interface has influenced the emphasis that will be placed on the nine functions of an ITS. Table I shows those relative emphases.

Table I. ITS Functions and Their Relative Emphasis

FUNCTION	FOCUS OF THIS PHASE	LEVEL OF IMPLEMENTATION	
		Simple	Complex
Initiating the tutoring session		●	
Assessing the student's status			●
Presenting the problem	●		●
Monitoring the student's performance		●	
Assessing the student's goal		●	
Identify the information to be tutored	●		●
Adapting tutor mode to student	●		●
Tutoring the student	●		●
Updating the student model		●	

The STOL ITS will represent STOL as a command language in its linear textual form in the context of the STOL interface as shown in Figure 2. In addition, the designers would like to develop at least one other form of STOL representation that would facilitate the learning and tutoring process. This would be a key feature of the ITS that would distinguish it from a simulator, just capable of representing the operational environment. The psychological motivation for this is that student's of programming languages can easily overload their working memory with syntax rules. One way to reduce this cognitive workload, is to provide the student with a structured editor such as the one in the LISP Tutor (Anderson & Reiser, 1985). With that editor, when the student enters a LISP function, the tutor displays place holders for the required arguments. The editor also automatically balances parentheses. This form of editor does reduce cognitive workload, but it fails to solve another problem common among programming language students, their difficulty in translating their natural language solutions into the narrow and restrictive environment of most programming languages. One solutions to this problem, proposed by Bonar et at. (1988), is to provide the student with some intermediate representation of the programming language that will be less taxing to the student's working memory and at the same time provides an easier transition from the normal way of structuring the problem.

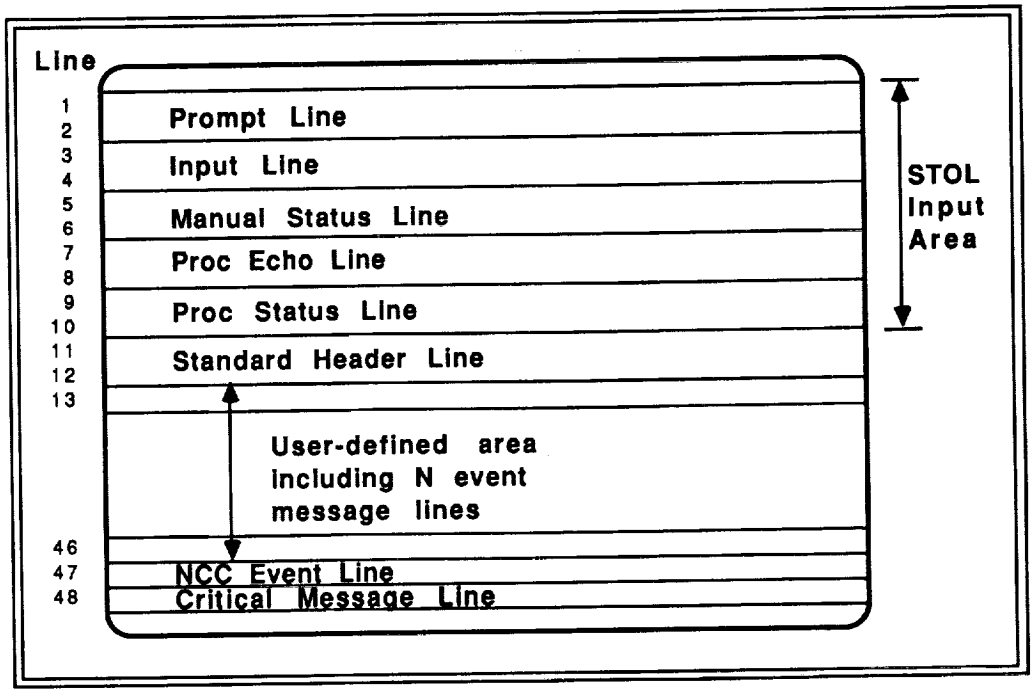

Figure 2. The STOL User Interface

In the case of Bonar et al. (1988), this intermediate representation took the form of a visual programming language. More precisely, they used a set of icons to represent the programming plans required to solve a limited set of Pascal programming problems. These icons were in the form of puzzle pieces emphasizing the placement of these plans in relation to other plans. A similar approach may be helpful for STOL students, and a several forms of visual programming languages and direct manipulation environments will be evaluated.

STOL was selected as the command language for this phase of the ITS implementation. STOL is used in a number of Payload Operations Control Centers (POCCs) to control ground system elements. It allows for the control of telemetry, command and display processing, and related support functions. There were a several key considerations in selecting a specific POCC. First, the POCC had to have a number of active controllers using STOL that would serve both as subject matter experts and as evaluators of the STOL ITS prototype. The POCC had to have a number of STOL controlled tasks that were similar to the majority of other POCCs, and the controllers had to be active and accessible during this phase of the ITS design.

The Gamma Ray Observatory (GRO) POCC met the above criteria and was selected for this phase of the ITS development. There are currently six to eight GRO operators, with that number increasing to 12 or 14 during critical activities such as the launch. These operators will be available to provide both the preliminary STOL data and the in-depth GRO task information. Once the prototype is developed, these operators will be able to provide problem solutions as well as ways of optimizing STOL representations.

The goal of the STOL tutor is to provide the critical training to controllers who need to learn the language. For the early stages of development, the STOL ITS will be limited to STOL as it is used in the GRO POCC, and as it is applied to a limited range of payload control problems. This is a similar approach as was taken in the development of PROUST (Johnson, 1986), a tutor that analyzes Pascal programs for non-syntactic bugs. During the development phase, PROUST was designed to analyze two programming problems. The developers demonstrated PROUST's ability to diagnose hundreds of novice solutions to these two problems. They wanted to establish PROUST's diagnostic robustness before expanding the number of problems that it could diagnose. During this phase of the ITS design, several GRO POCC problems will be identified for the tutoring process. These problems should be similar to problems solved through STOL at other POCCs. They should be problems which are relatively difficult for novices to solve and that require a range of STOL skills. Finally, these problems should be related to tasks that are critical to mission success.

The ultimate goal of this project is to develop a set of ITS modules with sufficient flexibility so they may be used for the tutoring of other command languages used at NASA control centers.

ACKNOWLEDGEMENTS

Financial support for this endeavor has been provided by CODE 522.3 of Goddard Space Flight Center. We would also gratefully acknowledge the help of Henry Murray, CODE 511.2 of Goddard Space Flight Center, in coordinating interviews with subject matter experts in the GRO POCC.

REFERENCES

Anderson, J. R. (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 Publishers.

Anderson, J. R., Boyle, C. F., & Yost, G. (1985). The Geometry Tutor. *Proceedings of the International Joint Conference on Artificial Intelligence*. Los Altos, CA: Morgan Kaufmann Publishers, Inc.

Anderson, J. R., & Reiser, B. J. (1985). The LISP tutor. *BYTE*, 10:4, 159-175.

Bonar, J., & Cunningham, R. (1988). BRIDGE: An intelligent tutor for thinking about programming. In J. Self, (Ed.) *Artificial Intelligence and Human Learning* (pp. 391-409). New York: Chapman & Hall Ltd.

Burns, H. L. and Capps, C. G. (1988) Foundations of Intelligent Tutoring Systems: An Introduction in *Foundations of Intelligent Tutoring Systems* Polson, M. C., and Richardson, J., (eds) Hillsdale, NJ: Lawrence Erlbaum Associates, Publishers.

Carroll, J. M., & McKendree, J. (1987). Interface design issues for advice-giving expert systems. *Communications of the ACM*, 30, 14-31.

Eike, D. & Seamster, T. L. (1989). Application of ITS technology to NASA control centers. Final Technical Report under Contract SEAS CAR-2911800.

Johnson, W. L. (1986). Intention-based diagnosis of novice programming errors. Los Altos: Morgan Kaufmann Publishers, Inc.

Norcio, A., Stanley, J. (1988). *Adaptive Human Computer Interfaces.* NRL Report 9148, Washington, D.C.: Naval Research Laboratory.

Wenger, E. (1987). *Artificial Intelligence and Tutoring Systems.* Morgan Kaufmann Publishers, Inc.

Woolf, B. (1988). 20 years in the trenches: What have we learned? *Proceedings of the International Conference on Intelligent Tutoring Systems.*