To appear in proceedings of Aero-HCI 2000, September 27th to 29th, Toulouse, France

Effective Strategies for Bridging Gulfs Between Users and Computer Systems

Santosh Mathan, Kenneth Koedinger, Albert Corbett, Arn Hyndman

Center for Innovation in Learning / HCI Institute, Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891
mathan@cmu.edu

ABSTRACT

This paper describes design strategies that led to significant improvements in the usability and learnability of an end-user programming environment called RIDES. These strategies may be viewed as concrete procedures for building easily learned interfaces and were derived from Polson and Lewis' CE+ theory.

RIDES was designed to make it possible for US Air Force training experts, with minimal programming experience, to author simulation-based Intelligent Tutoring Systems. Tutors created using RIDES facilitate guided learning-by-doing. Such an instructional approach has been validated as an effective way of acquiring complex procedural skills in various contexts. Instruction created using RIDES has the potential for addressing training problems associated with the operation of highly automated aviation systems.

Keywords

training, cognitive walkthrough, CE+, interface design

INTRODUCTION

Deficiencies in the knowledge and skill required to operate flight deck avionics have been highlighted as factors that compromise operational efficiency and safety [3]. Research indicates that these deficiencies, in part, stem from inadequate training. Training departments often expect a substantial amount of avionics specific skills to be acquired on the job [3]. While many skills may in fact be built and maintained in such a manner, many others may fail to be acquired. Resulting deficiencies may contribute to increased workload and impact efficiency and safety in certain operational situations.

The training concerns noted above are not unique to flight training, similar observations been made in ground-based aviation tasks that are supported by high levels of automation (e.g. Lesgold [5] et al., who describe issues associated with the training of F-15 avionics technicians).

Intelligent Tutoring Systems (ITS) represent an area of research aimed at addressing training problems. A web based autopilot tutor developed by Sherry et al is a recent example of such an effort. These systems allow learners to acquire skills by actually performing complex tasks likely to be encountered on the job. Context specific hints and error feedback keep problem solving productive and efficient by minimizing floundering and associated frustration. Such an instructional approach has been shown to be a very effective and efficient way of acquiring complex procedural skills in various training contexts [1].

The tutoring systems just described support guided learning-by-doing. At the heart of these systems is a cognitive model that represents the skills required to perform complex tasks. Cognitive models underlying these tutors are typically based on a systematic analysis of the skills required to perform complex tasks. These models serve as the basis for developing curriculum. Furthermore, they aid in the interpretation of student actions so as to provide context-specific assistance.

Intelligent Tutoring Systems have been successfully applied in the aviation domain. For example, an independent evaluation of SHERLOCK, a trouble shooting tutor, demonstrated that US Air Force F-15 avionics technicians, with about 20 to 25 hours of training on the system, were able to perform as well in trouble shooting avionics test equipment failure situations as colleagues with 4 years of on-the-job experience [5]. Despite such success, intelligent tutoring systems have not been widely adopted in aviation training contexts. One factor has been the expense involved in creating such systems — SHERLOCK took several programmer-years of development resources. Moreover, training experts often lack the substantial programming expertise required to develop such instruction.

RIDES [6], the system that forms the focus of the design effort reported in this paper, was designed to simplify the development of computer tutors. It was designed to allow subject matter experts in the US Air Force, with little or no programming experience, to develop interactive device simulations and associated tutorials. It has been used to create graphical simulations and tutorials associated with complex devices such as air traffic control panels, aircraft hydraulic circuits, and cockpit controls, to name a few.

Although the functionality offered by RIDES has the potential for addressing vital instructional needs within the military and industry, it has not been widely used. It has been speculated that usability problems have been a limiting factor in its adoption, particularly by subject matter experts with little programming experience. Thus, we undertook a design effort to improve the usability of the system.

In the following pages we describe concrete design strategies derived from Lewis and Polson's CE+ theory. We illustrate practical application of these strategies in the design of RIDES. Furthermore, we demonstrate the efficacy of our design by reporting on the results of two experimental studies.

Identifying Design Strategies

Norman has characterized problems involved in human interaction with computer systems as arising from two gulfs [7]. These are: the "gulf of execution", which separates a user's goals from the actions needed to accomplish them, and the "gulf of evaluation" which impedes appropriate

interpretation of system state. Bridging each of these gulfs calls for distinct design strategies — strategies for bringing the user closer to the system and those for bringing the system closer to the user. These gulfs are particularly pronounced in the case of end-user programming systems; programming environments are intrinsically complex and the entry characteristics of end-users often leave them unprepared to deal with such complexity.

One way to bridge the gulfs separating users from computer systems is to build systems that are amenable to learning by exploration. Polson and Lewis laid a foundation for the design of easily learned interfaces in their CE+ theory [8]. CE+ provides an account of exploratory learning of computer systems. Such an approach to developing proficiency exploits the fact that, in problem solving contexts, people learn best by doing [1]. CE+ assumes the following four premises about user interaction (which correspond with the four questions in a Cognitive Walkthrough [11]): 1) The user sets a goal to be accomplished with the system, 2) the user searches the interface for currently available actions, 3) the user selects action that seems likely to progress towards goal, 4) the user performs selected action and evaluates progress being made towards current goal.

Our approach was to consider each of the 4 premises of user interaction offered by CE+, and in association with design principles specified by Polson and Lewis [8], attempt to derive concrete interface design strategies.

User Sets Goals to Be Accomplished With the System

This premise of CE+ calls for a user interface that is congruent with likely goal structures held by system users. However, Polson and Lewis do not prescribe specific techniques for ascertaining goal structures and determining the extent to which user and system goal structures overlap. We adopted elements of the design approach suggested by Kieras & Polson [4]. Kieras & Polson illustrate formal usability techniques of writing production rules and goal trees to characterize user performance. In particular, constructing and comparing goal trees that characterize anticipated user expectations on one hand and the procedures implicit in a candidate design on the other. Such an approach provides the basis for structuring the system so that the temporal sequence of steps required of a user conform more closely to a user's goals and expectations.

User searches the interface for currently available actions and selects actions likely to make progress towards goal.

CE+ suggests that in the course of learning to use a system by exploration, interface elements such as prompts and buttons serve to create sub goals that become components of the overall task goal held by a user [9]. These goals typically involve selection of actions that will enable the user to make progress on the task. To facilitate the formation of appropriate goals and enable selection of appropriate actions, Polson and Lewis call for making available actions salient, using identity cues between actions and goals, making actions easy to discriminate, offering few action alternatives in specific goal contexts, and keeping sequences of action choices short [8]. The space of interpretations for enabling these suggestions in an interface is large. We made these principles concrete in our design by:

- Creating task oriented dialogs consolidate functionality necessary for accomplishing specific tasks and situating these to facilitate easy discovery.
- Organizing the spatial layout in our dialogs to parallel the temporal sequence of actions required to accomplish a task.
- Communicating the sequence of goals by redundantly coding the interface with verbal prompts
- Locating controls necessary for executing actions in close proximity to these prompts
- Explicitly stating the actions to be performed by users in specific goal contexts.

User performs selected action and evaluates progress being made towards current goal.

Feedback from the system following a user action serves to initiate the selection of error correction goals or new goals relevant to the completion of a task. Polson et. al recommend using identity cues that link responses and user goals and providing an obvious ways to undo actions [9]. In our implementation we provided feedback following actions by:

- Displaying the result of actions in close physical proximity to the goal prompt
- Making the area of the interface associated with the next goal in the sequence visually salient through means such as highlighting and change of focus
- Alerting users through descriptive error messages.

Later in this paper we illustrate the application of these strategies in the context of the design of two interfaces and compare these with the original system in two experimental user studies.

The Design Context

RIDES is an environment for authoring and delivering interactive computer-based instruction associated with complex devices [6]. Based on a few user-specified parameters, the system can automatically generate training lessons that are appropriate for device simulations. These exercises, known as "patterned exercises", teach students how to find device components, identify components, check for malfunctions, diagnose malfunctions, and manipulate controls to execute device procedures or place the device in particular states. These exercises can be played back to students in three modes. In demo mode, the system demonstrates a particular skill to students; practice mode allows students to practice skills with some guidance from the system. In test mode students are tested on skills embodied in the exercise. In addition to automatically generated patterned exercises, RIDES provides enhanced lesson authoring facilities for users to customize patterned exercises they have generated, or to craft novel exercises from scratch.

REDESIGN EXPERIMENT 1

Interface Description

Our design efforts were first directed at patterned exercise authoring in RIDES. Patterned exercise functionality allows authors to generate a variety of predefined exercise types.

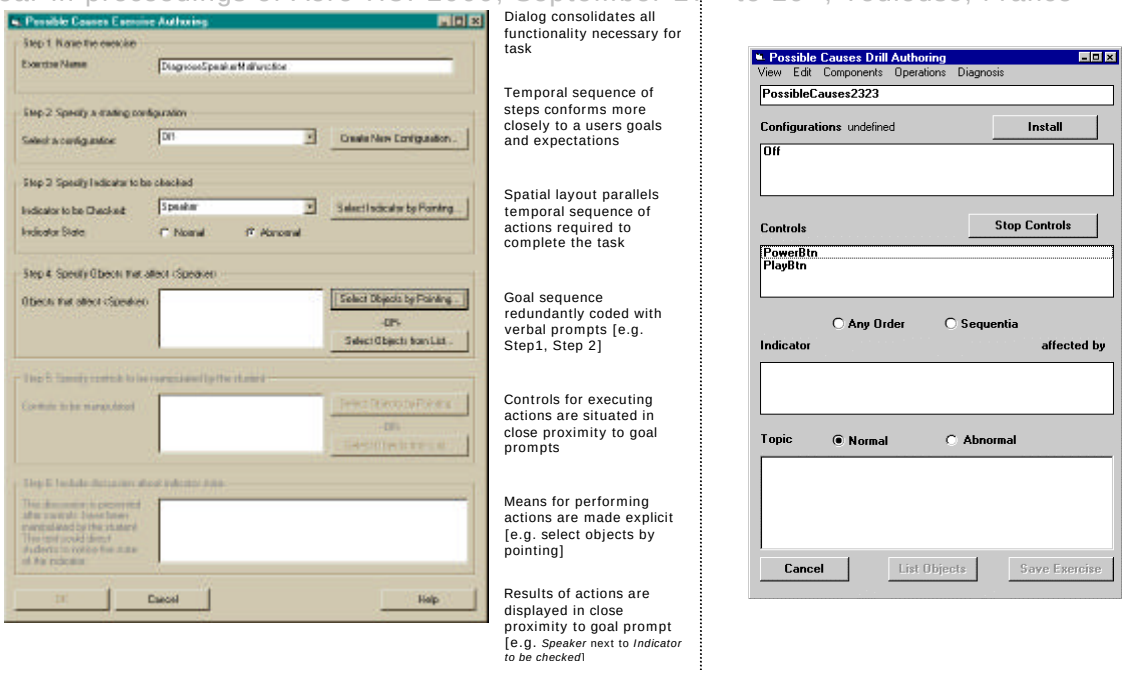

Figure 1: Redesigned Patterned Exercise Dialog (left) Original Patterned Exercise Dialog (right)

These include exercises for finding or naming components in a simulated device, performing device operation procedures, achieving device state goals (e.g., turning a CD player on) or finding the possible causes for a device fault. The steps for creating an exercise vary somewhat depending on the type of exercise. However, as one of the more complicated patterned exercises, the *Possible Causes Exercise* is representative of most of the steps involved in authoring any type of patterned exercise. A *Possible Causes Exercise* teaches students to check for, and diagnose problems with a device.

Figure 2a shows the key steps in *Possible Causes Exercise* authoring, organized in terms of the major goals that must be fulfilled. First, the author must create a "knowledge unit" that specifies causal relationships between objects that will be used in instruction about finding potential system faults. For instance, in the CD Player simulation an author could indicate that the SpeakerWire object affects the Speaker (this information is used in generating instruction about diagnosing why the Speaker isn't functioning). The second major goal is to create any "configurations" needed in instruction. Configurations are fixed states of the simulation (e.g., a CD Player with the power off and a bad speaker wire) that will play role in the exercise. The third major goal is to author the patterned exercise itself using the patterned exercise dialog (see Figure 1, right). This process involves a number of substeps -- as shown in the rightmost branch of Figure 2a.

We performed a goal tree analysis of the patterned exercise procedure, both as required in RIDES (Figure 2a), and as a novice user might expect (Figure 2b). The goal tree analysis revealed a fundamental problem with the RIDES interface. The organization of the RIDES interface often requires users to accomplish subtasks before creating a patterned exercise using the patterned exercise editor. For instance, in order to author a Possible Causes Exercise, users must create a configuration and knowledge unit before using the Possible Causes Exercise authoring interface. Such subtasks do not correspond to a user's likely prior conception of the steps needed to achieve their goal of creating an exercise.

Figure 1 (left) depicts a redesigned patterned exercise dialog. The following design revisions are evident:

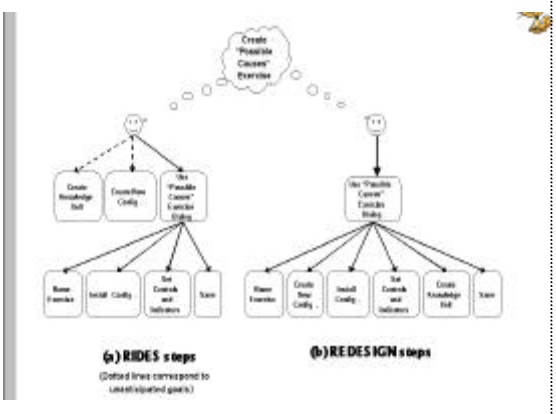

Figure 2: Steps in authoring a Patterned Exercise

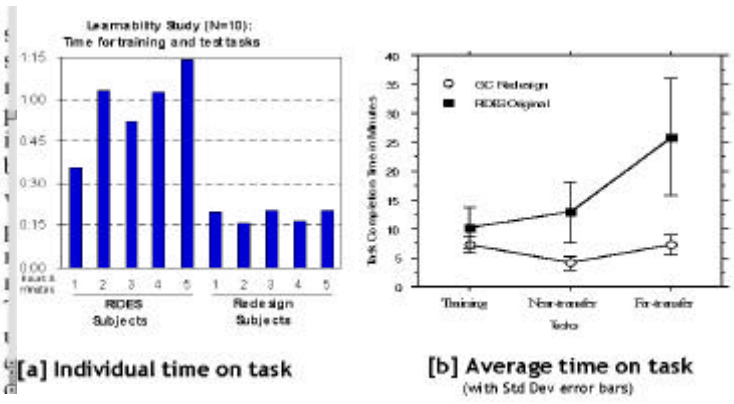

Figure 3: Time on task comparisons -- RIDES vs. Redesign

- Successive steps are labeled in this dialog and widgets associated with each step are clearly indicated within a border drawn around each step.
- Important system specific subtasks, such as creating a new configuration or knowledge unit, can be accomplished in the context of a task oriented interface with prompts from the system
- All actions required of the user is communicated using descriptive labels. Multiple ways of specifying information are presented to users.
- Highlighting and application focus changes provide feedback on the successful execution of actions. (e.g. Step 4 become active after Step 3)

Experimental Validation

We conducted a usability study to evaluate the efficacy of our solution. We implemented Visual Basic™ prototypes of the original and redesigned *Patterned Exercise* authoring interfaces. The original version of RIDES runs under the X windows environment. Our decision to implement a high fidelity mockup of the original system was an attempt to control for the possible influence of the window manager on user performance.

Subjects

Fourteen Carnegie Mellon University students participated as paid volunteers. Each subject used one of the two patterned exercise authoring designs (RIDES or Redesigned Interface) to author exercises associated with a CD player simulation. Like intended RIDES users, all subjects were computer literate, and experts in the simulation domain. The first four participants acted as pilot subjects. Pilot evaluations allowed us to make iterative improvements to the redesigned interface, training procedure and other test materials. Of the remaining ten subjects, five worked with RIDES and five with the Redesign prototype.

Procedure

After a reading a brief handout describing some basic Microsoft Windows 95™ concepts, test participants walked through a tutorial document designed to introduce users to some fundamental concepts in RIDES. The document facilitated guided exploration of the interface. Following the tutorial, test participants performed three authoring tasks. Users were asked to create a *Goal Exercise* (exercise designed to teach students to accomplish the goal of playing a CD on a CD player), *Find Exercise* (exercise designed to show students how to find various CD player components), and a *Possible Causes Exercise* (exercise designed to teach students to identify possible causes for a speaker failure). The first authoring task was a training task; subjects followed step-by-step instructions in authoring the Goal Exercise. The remaining two tasks tested user ability to transfer skills acquired to authoring patterned exercises on their own. A two-hour time limit was imposed for completing the three exercises.

Results

The total time required for subjects to complete the training task and two transfer tasks was 49.1 minutes on average for the RIDES Original subjects and 18.7 minutes on average for the redesign subjects. Thus, the redesign cut total training and transfer time down to less than a half, nearly one third (38%), the RIDES time. As Figure 3a (previous page) depicts, there was no overlap in performance times between subjects in the two groups. The slowest Redesign subject (20.8 minutes) was still faster than the fastest RIDES subject (31.8 minutes).

Although the formal comparison portion of the user study involved only 10 subjects, the differences were large enough and consistent enough to be statistically significant. We performed two way ANOVA with Interface Condition (RIDES Original vs. redesign) as a between subjects factor and Authoring Task (Training vs. Near Transfer vs. Far Transfer) as a within-subjects factor. Figure 3b (previous page) illustrates the two Condition means across the three tasks. The main effect of Condition was statistically significant ($F(1, 8) = 27.39$, $p < .001$) confirming the large observed difference described above in favor of the redesign. The main effect of Task was also statistically significant ($F(2, 16) = 9.22$, $p < .01$) with the Far Transfer task requiring significantly longer than the other two. Interestingly, the interaction of Condition and Task was also significant ($F(2, 16) = 6.94$, $p < .01$). As can be seen in Figure 3b, there was no difference in the Training task where subjects in both conditions executed steps described in walkthrough provided in the manual ($F(1,8) = 3.00$, $p > .10$). However, already on the Near Transfer task, a significant difference emerges ($F(1,8) = 13.04$, $p < .01$). Subjects using the RIDES Original interface had trouble both in remembering what they had learned and in rediscovering new or forgotten action procedures. In contrast, subjects using the redesign interface had little trouble transferring what they had learned or rediscovering action procedures as needed. The differences on the Far Transfer task are even more dramatic ($F(1,8) = 16.90$, $p < .01$). Subjects using the RIDES Original interface floundered considerably. However, subjects using the redesign progressed smoothly performing the greater number of actions in the Far Transfer task at about the same rate they had done so in the Near Transfer task.

When we look at the data on the number of experimenter interventions we find a similar pattern of results. The experimenter intervened when the subject asked for help or when the subject made a critical error and was unable to correct it within one minute. Across the three tasks, the experimenter intervened an average of 2.4 times in the Redesign condition and 8.2 times in the RIDES condition ($F(1, 7) = 5.3$, $p = .08$).

REDESIGN EFFORT II

The success of our in the context of the *Patterned Exercise* interface might be attributed to the relative simplicity of the functional requirements for *Patterned Exercise* creation. However, we believe the method is also applicable for more open-ended tasks and systems with substantial functional complexity. To demonstrate this claim, we applied our design strategies to design a better interface for RIDES *Custom Lesson Authoring* capabilities.

Interface Description

Execution of any RIDES lesson consists of a series of display actions performed by RIDES (e.g. install configuration, display text, highlight object) followed by a student response (click anywhere, enter text, click done button). Patterned Exercise functionality automatically specifies details of the instructional interaction for certain

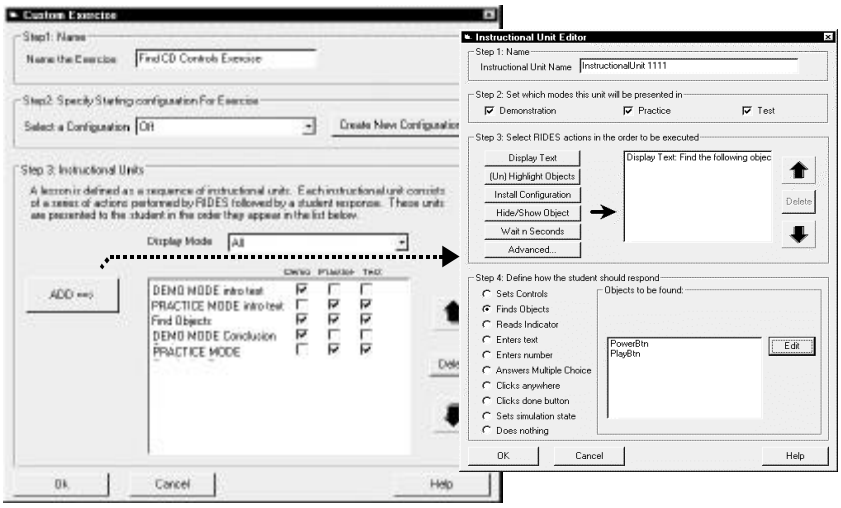

Figure 4: Redesigned Custom Authoring Editor

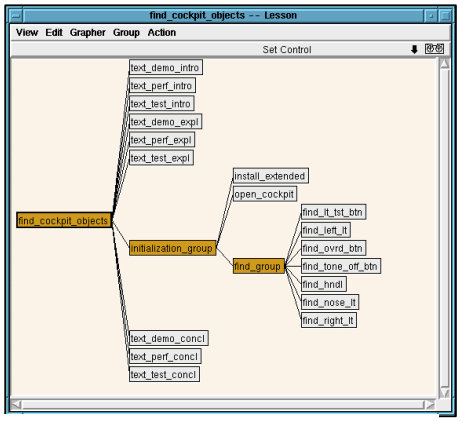

Figure 5: RIDES Custom Authoring

types of lessons. RIDES provides the so-called *Custom Lesson Authoring* interface for users to modify patterned exercises they may have generated, or to craft novel exercises from scratch (see Figure 5). *Custom Lesson Authoring* functionality provides users with a powerful and flexible way of creating interactive training simulations; the functionality offered by the system is comparable to multimedia authoring tools like Macromedia Director, Hypercard etc.

In creating a *Custom Lesson*, the author's task is to specify information that is to be displayed to students and to indicate the appropriate student response. The author must also express the conditional and sequential relationships between interaction elements that make up the lesson. RIDES *Custom Authoring* functionality requires users to specify these details in the form of a hierarchical representation by employing a variety of widely scattered menu options (Figure 5). Specifying instructional intent in this manner might not always be a straightforward task for users.

Evaluation and Redesign

As in the earlier redesign study, our goal was to create an easily learnable interface. We developed an alternate representation that corresponds closely to the actual execution of a lesson.

Each RIDES lesson is a sequence of one or more system actions (such as displaying text, highlighting objects, setting a configuration etc.) followed by a user response -- we began referring to each system action and user response pair as an "instructional unit". Specifying lessons as a series of instructional units would be more natural for users as it closely paralleled the actual execution of the lesson.

As the dialog on the left in Figure 4 indicates, a two dimensional representation is employed to distinguish between temporal and conditional relationships. Successive "instructional units" are displayed from top-to-bottom in the instructional unit window. Three columns of check boxes represent exercise mode conditionality. Each instructional unit is structured as a RIDES action and optional student response. Furthermore, we employed the strategies displayed in Figure 1 in the design of these dialogs.

Experimental Validation

We conducted an experiment to evaluate the effectiveness of the new custom lesson authoring design.

Subjects and Procedure

Eight subjects completed three authoring tasks in this study. Four subjects worked in the original RIDES custom authoring interface and four worked with the redesigned interface. As in the earlier study, the redesign specifications were programmed in visual basic. The three authoring tasks from the first empirical usability study of patterned exercise authoring were again employed in this study. In the first task, subjects worked from detailed step-by-step instructions in authoring a Goal Drill. In the remaining two tasks students were only presented overall exercise objectives. The second task was a Find Drill and the third task was a Possible Causes Exercise. A two-hour time limit was imposed for completing the three exercises.

Results

Overall, students in the Redesign condition were able to complete the three tasks more quickly, as in the previous study. Three of the four subjects in the redesign condition were able to complete the three authoring tasks in the two-hour time period, while only one of the subjects in the original RIDES condition was able to finish the three tasks in two hours. Since all subjects completed the first two tasks, completion times for these two tasks can be directly compared. As we observed in the first experiment, there was no reliable difference between the two groups on the first task in which detailed instructions were provided. However, subjects working with the original RIDES interface required about 50% more time to complete the second task. The redesign prototype group averaged 15.6 minutes to complete this task while the RIDES group averaged 23.4 minutes. This difference is reliable at the .05 level. The custom authoring redesign was so successful that students working in this condition completed this second task almost as quickly as students using the far more structured RIDES patterned exercise authoring interface in

the previous study (15.6 minutes for the Redesign condition in this study vs. 12.9 minutes for the RIDES condition in the previous study).

Students in the RIDES authoring condition not only required more time to complete the tasks, but also appeared to require more experimenter interventions. As in the first study, the experimenter intervened only when the subject asked for help or when the subject made a critical error and was unable to correct it within one minute. In tasks 2 and 3, the experimenter intervened an average of 4.75 times in the redesign prototype condition and 8 times in the RIDES condition. This difference is marginally significant, $p = .08$.

DISCUSSION

Research suggests that flight crews are often deficient in the skills associated with the operation of flightdeck avionics [3]. These deficiencies can contribute to increased workload and may compromise operational efficiency and safety in certain situations. Research points to training practices as a source of these problems [3].

Efforts aimed at addressing training problems have included the development of Intelligent Tutoring Systems [e.g. 10]. These systems facilitate guided learning-by-doing and have been applied quite successfully in aviation training contexts [5]. However the cost and technical difficulties of developing these systems have contributed to limited adoption of within the aviation community.

RIDES, the focus of the design effort reported in this paper, was designed to enable training experts, with minimal programming experience, to develop intelligent tutors. However usability problems have come in the way of achieving this goal. Our design effort sought to address these problems by constructing an interface that was amenable to learning by exploration

Polson and Lewis' CE+ theory [8] offers a basis for the design of systems that can be learned by exploration However, the space of possible interpretations one could draw from the theory and associated design principles can be quite large. What we have done in this paper is to derive a set of concrete design strategies from CE+ and applied them in the design context of RIDES. The experiments reported here demonstrate that these strategies enable exploratory learning even in fairly complex applications and not just in the simple walk up and use interfaces that CE+ theory and cognitive walkthrough have commonly been identified with. The strategies applied to the redesigned interfaces in this paper are listed in Figure 6.

Strategies for bringing the system closer to the user

- Construct goal trees to characterize user expectations And procedures implicit in candidate design
- Organize system so that the system goal structures Conform to anticipated user goal structures

Strategies for bringing the user closer to the system

- Create task oriented dialogs that consolidate controls for performing specific tasks and situate these so as to facilitate easy discovery
- Organize spatial layout in dialogs to parallel the temporal sequence of actions required to accomplish task
- Communicate appropriate sequence of steps by redundantly coding the interface with verbal prompts (e.g. "Step 1", "Step 2")
- Explicitly communicate actions required to be performed by users in specific goal contexts
- Provide controls for executing actions in close proximity to goal prompts
- Provide feedback on results of action in close proximity to goal prompt.
- Make area of interface associated with next goal in task sequence visually salient through means such as highlighting, and change of focus.
- Allow users to recover from errors

Figure 6: Design Strategies

Tradeoffs

While the application of the design strategies listed here may lead to functionality that is sufficient to carry out particular tasks, there is no guarantee that these will be the most efficient way to carry out them out (in the context of RIDES, however, a Keystroke Level Model [2] predicted shorter expert execution times for the redesigned dialogs). Moreover, the effectiveness of dialogs constructed using the design strategies mentioned here is closely related to the representation and decomposition of a user's goals. It may not be a trivial matter to characterize many tasks in such a way.

ACKNOWLEDGMENTS

This research was funded by the Office of Naval Research (grant N000149510771). The authors acknowledge Doug Brams, Ning Fan, and Angela Jury for their contributions to this redesign and evaluation. Thanks to Peter Polson for comments on a draft of this paper.

REFERENCES

1. Anderson, J. R. (1993). *Rules of the Mind*. Hillsdale, NJ: LEA.
2. Card, S.., Moran, T., & Newell, A. (1983). *The Psychology of Human-Computer Interaction.* Hillsdale, N.J.: LEA
3. FAA (1996). The Interface Between Flightcrews and Modern Flightdeck Systems. (http://www.faa.gov/avr/afs/interfac.pdf)
4. Kieras, D. & Polson, P. G. (1985). An approach to the formal analysis of user complexity. In International Journal of Man Machine Studies, 22, 365-394.
5. Lesgold, A. M., Lajoie, S. P., Bunzo, M., and Eggan, G. (1993). SHERLOCK: A coached practice environment for an electronics troubleshooting job. In J. Larkin, R. Chabay (Eds.), Computer assisted instruction and intelligent tutoring systems. Hillsdale, NJ: LEA
6. Munro, A. (1994). RIDES Authoring Reference. Behavioral Technology Laboratories, USC (http://btl.usc.edu/rides/)
7. Norman,D.A. (1986). Cognitive engineering. In D.A. Norman & S.W. Draper (Eds.), User centered systems design: (pp31 - 61). Hillsdale, NJ: LEA
8. Polson, P.G., Lewis,C, Theory-based design for easily learned interfaces. HCI, 5, 191-220, 1990.
9. Polson, P., Lewis, C., Rieman, J., & Wharton, C. (1992). Cognitive walkthroughs: A method for theory- based evaluation of interfaces. International Journal for Man-Machine Studies, 36, 733-741.
10. Sherry, L., Feary, M., Polson, P., Palmer, E., Autopilot Tutor: Building and Maintaining Autopilot Skills. (In Press)
11. Wharton, C., Rieman, J., Lewis, C. and Polson, P. (1994). The cognitive walkthrough method: A practitioner's guide. In J. Nielsen & R.L. Mack (Eds.) Usability Inspection Methods. pp. 105-140. New York: John Wiley & Sons.