

# Theory-Based Design for Easily Learned Interfaces

by

Peter G. Polson and Clayton H. Lewis

Institute of Cognitive Science

University of Colorado

Campus Box 345

Boulder, CO 80309-0345

ICS Technical Report #88-16

---

## Theory-Based Design for Easily Learned Interfaces

Peter G. Polson and Clayton H. Lewis


*Institute of Cognitive Science, University of Colorado*

---

---

A preliminary version of this paper was presented at the Winter Conference of the Human-Computer Interaction Consortium in Vail, Colorado, January 22-24, 1989.

The authors made equal contributions to this paper. Polson lost the argument about the order of authorship. The authors' present addresses: Peter Polson, Department of Psychology, Campus Box 345, University of Colorado, Boulder, CO 80309; Clayton Lewis, Department of Computer Science, Campus Box 430, University of Colorado, Boulder, CO 80309



## Abstract

Many important computer applications require that users be able to make effective use of them with little or no formal training. Current examples include bank teller machines and airport information kiosks. Today, successful systems of this kind can only be developed by iteration using costly empirical testing. This paper aims to provide a theoretical foundation for the design of such systems, a model of learning by exploration called CE+. The theory incorporates assumptions from 1) the GOMS model and Cognitive Complexity Theory on the representation of procedural knowledge as productions, 2) the EXPL model on learning from examples, and 3) research on problem solving processes for simple puzzle-like problems. Design guidelines for systems that can be learned by exploration, Design-For-Successful-Guessing, are derived from the theory. These principles are compared to those developed by Norman (1988).

## TABLE OF CONTENTS

### 1. INTRODUCTION

#### 1.1. The Problem

#### 1.2. A Solution

### 2. REPRESENTING AND QUANTIFYING PROCEDURAL KNOWLEDGE

#### 2.1. The GOMS Model

##### A GOMS Example

##### Performance Predictions

#### 2.2. Cognitive Complexity Theory (CCT)

##### A CCT Example

##### Learning, Transfer, and Performance Predictions

#### 2.3. The Use of GOMS and CCT Models in Design

### 3. LEARNING PROCEDURES FROM EXAMPLES: THE EXPL MODEL

#### 3.1. An EXPL Example

#### 3.2. Use of EXPL in Design

#### 3.3. Limitations of EXPL

### 4. PROBLEM-SOLVING IN UNFAMILIAR DOMAINS

#### 4.1. Problem Spaces

#### 4.2. Search Methods

##### Hill Climbing

##### Back Chaining

#### 4.3. Label-Following

### 5. RELATING PROBLEM-SOLVING AND LEARNING

### 6. THE CE+ MODEL

#### 6.1. The Problem-solving Component of CE+

#### 6.2. The Learning Component of CE+

#### 6.3. The Execution Component of CE+

#### 6.4. A Related Model

#### 6.5. A Detailed Example

### 7. DESIGN FOR SUCCESSFUL GUESSING

### 8. NORMAN'S ACTION MODEL AND DESIGN GUIDELINES.

### 9. DISCUSSION

## 1. INTRODUCTION

This paper proposes a cognitive theory of initial learning in human-computer interaction and an associated design framework to support development of systems that require minimal learning on the part of users. Recent advances in cognitive and machine theories of learning, combined with ideas from earlier work on problem-solving, offer the hope that principled design of such systems may be possible.

### 1.1. The Problem

Many important computer applications, actual and potential, depend upon holding training requirements for users to a minimum. So-called walk-up-and-use applications aim to permit users to do useful work on their first encounter, without any organized preparation. Examples in use today include bank teller machines and airport information kiosks, and it is easy to envision a wide variety of future applications delivering information services to occasional users who will make no investment in learning to operate the systems concerned.

### 1.2. A Solution

Creating such systems today is an art, relying on intensive empirical testing without a clear principled basis for design. This paper presents a theoretical model of learning by exploration, the process users engage in in using an unfamiliar system, that aims to provide such a basis.

The model, called CE+, builds on earlier theoretical work in human-computer interaction that treats aspects of the problem of learning by exploration. Specifically, it incorporates ideas from GOMS (Card, Moran, and Newell, 1983) and Cognitive Complexity Theory (Kieras and Polson, 1985; Polson, 1987) on the representation of procedural knowledge, and ideas from EXPL (Lewis, Casner, Schoenberg, and Blake, 1987; Lewis, 1988) on how procedures can be learned from examples. To these it adds ideas adapted from

the literature on classical problem-solving about the control of exploration in unfamiliar domains (Polson and Jeffries, 1982; Greeno and Simon, 1988). The interaction between exploration, analysis, and learning in the resulting model is similar to that seen in ACT\* (Anderson 1983,1987) and SOAR (Laird, Newell, and Rosenbloom, 1987).

The presentation begins with pertinent ideas borrowed from earlier work in human-computer interaction, indicating what parts of the overall process of learning by exploration are dealt with and not dealt with by this existing work. The next section adapts ideas from problem-solving to fill some of the gaps in existing models. Then the complete model is described, with an illustration of its functioning in an actual task setting. The final section develops the implications of the model for the design of systems which can be effectively learned, and compares the resulting design principles, called Design-for-Successful-Guessing, to those of Norman (1988).

## 2. REPRESENTING AND QUANTIFYING PROCEDURAL KNOWLEDGE

### 2.1. The GOMS Model

In making a system easier to learn one could begin by trying to reduce the *amount* of knowledge that is required. The GOMS model (Card, Moran, and Newell, 1983) provides a starting point: it characterizes the knowledge necessary to make effective, routine use of software tools like an information service, a text editor, or a data-base manager. The GOMS acronym abbreviates the four categories into which this knowledge can be placed: *Goals*; *Operations*, which are the actions available in using the system; *Methods*, which are useful combinations of operations; and *Selection rules*, which specify how to choose an appropriate method for accomplishing a given goal.

#### A GOMS Example

A GOMS analysis involves writing a collection of methods that accomplish a task. Each method accomplishes a goal. The top-level method for

accomplishing the task decomposes it into a sequence of subgoals, and these subgoals may be further decomposed. The decomposition terminates in a sequence of operations that accomplish a goal or subgoal. The operations are the primitive mental operations defined by the theory and user actions, e.g., using a mouse to move the cursor, that the analyst chooses not to decompose into more primitive operations. Kieras (1988) has described a methodology for doing GOMS analyses and proposes a specific notation.

---

**Figure 1. A GOMS Analysis of the Task of Checking Spelling for A Menu Based Word Processor**

Method to accomplish goal of checking spelling

Step 1: Remember ID-LETTER is D; Remember Next Menu is SPELLING-TASK-MENU;  
Accomplish Goal Go-To Next- Menu.

Step 2: Verify Transition to SPELLING-TASK-MENU.

Step 3: Remember ID-LETTER is A; Remember Choice is CHECK DOCUMENT;  
Accomplish Goal Select Item-From- Menu.

Step 4: Accomplish Goal of Enter Document-Name.

Step 5: Accomplish Goal of Enter Diskette-Name.

Step 6: Verify Spelling-Check Complete.

---

Figure 1 is a GOMS analysis, in Kieras's notation, of the method for checking spelling on an IBM DisplayWriter for an early version of the software. The system is a stand-alone, floppy-disk-based word processor with a user interface employing full screen menus. The task requires the user to make two menu choices and enter document and diskette names.

The example shown is abbreviated. This method is called by a higher-level method, not shown, that reads task instructions and calls, using selection

rules, the appropriate method to perform the task. The example also omits lower-level methods, called by the check-spelling method, for making menu transitions, selecting items from menus, and entering document and diskette names.

A whole family of different GOMS model for this task could be built by varying the grain size of the analysis, that is, the level at which operations are not further subdivided in the analysis. The version shown in Figure 1 is at the most detailed grain size explored by Card, Moran, and Newell (1983).

### **Performance Predictions**

Card, Moran and Newell developed the GOMS model to make performance predictions for expert users carrying out routine tasks. They assumed that each operation has a fixed execution time and that execution time can be calculated by simply summing up the individual operation times for the sequence.

While the GOMS notation was intended to describe the content and structure of the knowledge necessary to perform a task, its creators did not claim that the notation identified units of knowledge that could be quantified or compared across tasks in a simple manner. Accordingly, they did not make quantitative predictions about training time for tasks or about transfer of training between tasks.

## **2.2. Cognitive Complexity Theory (CCT)**

Kieras and Polson (1985) proposed that the knowledge represented in a GOMS model be formalized as a production system. They had two objectives. The first was to express the knowledge in GOMS models in a notation that would enable them to make more direct links with research and theory on the acquisition and performance of cognitive skills (e.g., Anderson, 1983). The second goal was to be able quantify the knowledge in a model so as to training and transfer of training. Their extension of GOMS is called Cognitive Complexity Theory, or CCT.



## A CCT Example

A CCT model is derived by first performing a GOMS analysis and then writing production rules that implement the methods and control structures described in the GOMS model. Kieras (1988) designed his notation for GOMS so that with a few exceptions each statement maps into a production rule. The method shown in Figure 1 has been rewritten in production form in Figure 2.

-----

**Figure 2. Production System Model for The Method for Checking Spelling**

```
(StartChkSpIllDoc
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (NOT (NOTE CHECKING-SPELLING DOCUMENT)))
  THEN
    ((Add STEP GO-TO SPELLING-TASKS-MENU)
     (Add NOTE CHECKING-SPELLING DOCUMENT)))

(ChkSpIllDoc.P1
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (STEP GO-TO SPELLING-TASKS-MENU)
      (SCREEN TASK-SELECTION-MENU))
  THEN
    ((Delete STEP GO-TO SPELLING-TASKS-MENU)
     (Add GOAL GO-TO NEXT-MENU)
     (Add NOTE ITEM IS SPELLING-TASKS)
     (Add NOTE ID-LETTER D)
     (Add STEP VERIFY TRANSITION-TO-SPELLING-TASKS-MENU)) )

(ChkSpIllDoc.P2
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (STEP VERIFY TRANSITION-TO-SPELLING-TASKS-MENU)
      (NOTE TRANSITION DONE)
      (SCREEN SPELLING-TASK-MENU))
  THEN
    ((DW.VerifyOperator)
     (Delete NOTE TRANSITION DONE)
     (Delete STEP VERIFY TRANSITION-TO-SPELLING-TASKS-MENU)
     (Add STEP SELECT CHECK-SPELLING)))
```

**Figure 2. Production System Model for The Method for Checking Spelling (Continued)**

```

(ChkSpIllDoc.P3
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (STEP SELECT CHECK-SPELLING)
      (SCREEN SPELLING-TASK-MENU))
  THEN
    ((Delete STEP SELECT CHECK-SPELLING)
     (Add GOAL SELECT ITEM-FROM-MENU)
     (Add NOTE ITEM IS CHECK-SPELLING)
     (Add NOTE ID-LETTER A)
     (Add STEP ENTER DOCUMENT-NAME)))

(ChkSpIllDoc.P4
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (STEP ENTER DOCUMENT-NAME)
      (NOTE ITEM SELECTED)
      (SCREEN PROMPT ENTER DOCUMENT-NAME))
  THEN
    ((Delete NOTE ITEM SELECTED)
     (Delete STEP ENTER DOCUMENT-NAME)
     (Add GOAL ENTER DOCUMENT-NAME)
     (Add NOTE DOCUMENT IS THE DOCUMENT)
     (Add STEP ENTER DISKETTE-NAME)))

(ChkSpIllDoc.P5
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (STEP ENTER DISKETTE-NAME)
      (NOTE ENTER-DOCUMENT-NAME DONE)
      (SCREEN PROMPT ENTER DISKETTE-NAME))
  THEN
    ((Delete NOTE ENTER-DOCUMENT-NAME DONE)
     (Delete STEP ENTER DISKETTE-NAME)
     (Add GOAL ENTER DISKETTE-NAME)
     (Add NOTE DISKETTE IS THE DISKETTE)
     (Add STEP VERIFY SPELLING-CHECK-COMPLETE)))

(ChkSpIllDoc.P6
  IF ((GOAL CHECK-SPELLING DOCUMENT)
      (STEP VERIFY SPELLING-CHECK-COMPLETE)
      (NOTE ENTER-DISKETTE-NAME DONE)
      (SCREEN SPELLING CHECK COMPLETE. WORDS MARKED:0))
  THEN
    ((DW.VerifyOperator)
     (Delete STEP VERIFY SPELLING-CHECK-COMPLETE)
     (Delete NOTE ENTER-DISKETTE-NAME DONE)
     (Add STEP FINISH CHECK-SPELLING-DOCUMENT)))

```

Each production begins with a label, which is not functional. The condition is labeled with IF and is a list of clauses that must be contained in working memory for the rule to execute (fire). The clauses specify a goal (GOAL . . .) or subgoal to execute a given step in the method, (STEP . . .), information that must be in working memory, (NOTE . . .), and screen contents like prompts, (SCREEN . . .). The action, prefaced by THEN, adds and deletes clauses from working memory and executes primitive operations, few of which appear in this fragment. Control is passed to a method by putting its parameters in working memory and then adding a goal to working memory to perform the method. Detailed descriptions of the control structure, templates for methods, and style rules for the models are in Bovair, Kieras, and Polson (1988).

### **Learning, Transfer, and Performance Predictions**

Following Anderson (1983), Kieras and Polson assumed that the rules are cognitive units. Thus, they argued in a manner analogous to Kintsch (1974) that these units should be equally difficult to learn. This permits the time to learn a method to be predicted simply by counting the number of productions in its definition and multiplying by the time needed to learn a production. Kieras and Polson also developed a theory of transfer closely analogous to Thorndike's common elements theory of transfer (Thorndike & Woodward, 1901). Kieras and Polson assumed that identical steps and methods common to several tasks are represented by shared productions. When these common productions are acquired earlier they are transferred without any cost into the representation of a new task. Thus, a user only has to acquire the productions that are unique to a new task. This transfer mechanism also provided them with an analysis of what is meant by consistency (Polson, 1988).

CCT makes performance predictions that are identical to those of the GOMS model. Like the GOMS model, it assumes that the execution time for a task is calculated by summing up the times necessary to execute the sequence of operations necessary to perform the task contained in the actions of the various productions plus a constant activation time for each rule.

Numerous studies have tested and confirmed these assumptions about learning and transfer; see Polson (1987, 1988). The transfer assumptions appear to hold quite generally. However, the assumption that productions are learned at a stable rate appears to hold only for constrained learning conditions, not typical of those which would hold for new users of an application.

The training procedures used in studies which have confirmed CCT's learning predictions prevents learners from executing incorrect steps in a procedure. Each step is checked by a tutoring system, and learners are asked to try again if the correct step is not made. After a set number of failed attempts the learner is told the correct step. This procedure limits the impact of difficult steps not only by cutting off exploration of the incorrect alternatives at any step but also by avoiding the often serious consequences of actually executing an incorrect step.

Using this procedure Kieras, Polson, and their colleagues have found training times per rule ranging from 20 to 40 seconds in various studies involving a wide variety of tasks and systems. However, studies of more natural learning conditions show that learning times are much more variable: seemingly minor flaws in the details of a user interface can completely derail the learning process, so that the time to discover and encode a given step in an interaction method can range from a few seconds to infinity. (Karat, Boyes, Weisgerber, and Schafer, 1986; Carroll and Rosson, 1987.)

These difficulties are not unexpected. The CCT predictions of learning times are based only how much material must be learned, and do not take into account how the material must be learned. The adequacy or inadequacy of prompts and feedback provided by a system, which clearly influence learning time under natural conditions, are not included in the CCT analysis.

### 2.3. The Use of GOMS and CCT Models in Design

Consistent with the thrust of the GOMS and CCT models, all modern design guidelines and principles that address cognitive issues implicitly (Smith and Mosier, 1986; Rubenstein and Hersh, 1984) or explicitly (Norman, 1986,1988; Hutchins, Hollan, and Norman, 1985; Card and Newell, 1985; Kieras and Polson, 1985; Polson, 1987; Bennett, Lorch, Kieras, and Polson, 1987) assume that the better design is the one that minimizes the amount and complexity of the new knowledge necessary to effectively use an application.

Bennett, Lorch, Kieras, and Polson (1987), and Karat, Fowler, and Gravelle (1987) draw similar conclusions about the utility and feasibility of the use of GOMS- and CTT-style theoretical models in the design process. Models are useful in gaining an understanding of any global training and performance differences between two competing designs. A fine-grained analysis can be used to drive an iterative design process focusing on the exact change needed to reduced training time and improve productivity.

However, as just noted, the GOMS and CCT analysis addresses only one factor influencing learning demand, namely the amount of knowledge required. It is clear that such other factors as the clarity and adequacy of prompts and feedback also influence learning demands, and we therefore need a theoretical framework account that deals with these.

### 3. LEARNING PROCEDURES FROM EXAMPLES: THE EXPL MODEL

Lewis and his colleagues (Lewis, Casner, Schoenberg, and Blake, 1987) have developed a model that attempts to account for the role of feedback in learning procedures. The model, called EXPL, uses a causal analysis of a sequence of user actions and system responses to determine which actions produce which responses, and hence to provide the basis for accomplishing novel tasks.

---

An analysis EXPL starts from an encoding by the theorist of the actions performed by a user and the responses produced by the system. These encodings break actions and responses down into smaller elements among which causal connections may be found. A few very general heuristics are then used to place causal links. For example, when an action and a later response share elements the *Identity heuristic* places a causal link between them: this heuristic deals with cases in which some user action that involves an object, say a picture on the screen, triggers a system response that involves the same object, say deleting the picture. Lewis (1986, 1988) describes the model in more detail.

### 3.1. An EXPL Example

Figure 3 shows an EXPL analysis of the check spelling method used in the GOMS (Figure 1) and CCT (Figure 2) examples presented earlier. Figure 4 shows the two menus used in the task. Let us look first at the analysis of the first user action and its context. Some of the work of analysis is already done in the encoding. Rather than working from a description of the user's physical action, typing 'd', the analysis starts from a description that indicates what this action does in the context in which it occurs: "choose \*spelling-tasks". The asterisk in the encoding marks an item which must appear on the screen before the user can act on it. In this example, the user cannot choose spelling tasks unless the corresponding item actually appears on the screen. The asterisk leads to the placement of a *prerequisite link* between "choose \*spelling-tasks" and the system action which placed \*spelling-tasks on the screen, s1. The prerequisite link indicates that one cannot perform the action "choose \*spelling-tasks" out of context; rather, preparatory actions may be needed to make \*spelling-tasks available.

When "choose \*spelling-tasks" is followed by the display of a menu containing the heading "spelling-tasks" and other items, the analysis seeks to attribute the appearance of each item to some prior user action. Since \*spelling-tasks occurs both in the description of the user action "choose \*spelling-tasks" and in the description of the system response, the Identity

Figure 3. EXPL Analysis of Spelling Check Interaction

<u>Events</u>	<u>EXPL encodings and causal links</u>	<u>Remarks</u>
TASK SELECTION menu (see top of Figure 3) on screen.	s1: show *task-selection *typing-tasks ... *spelling-tasks	Prerequisite link
User types 'd'	u1: choose *spelling-tasks	Link to *spelling-tasks placed by Identity heuristic; others by Previous Action heuristic
SPELLING TASKS menu appears (bottom of Figure 3)	s2: show *spelling-tasks *check-document ... *task-selection	Prerequisite link
User types 'a'	u2: choose *check-document	Links placed by Previous Action heuristic
System replaces prompt with "Type document name; press ENTER:"	s3: req *document-name	Prerequisite link
User types "security"	u3: type *document-name security	Links placed by Previous Action heuristic
System replaces prompt with "Type diskette name; press ENTER:"	s4: req *diskette-name	Prerequisite link
User types "able"	u4: type *diskette-name able	Links placed by Previous Action heuristic
Diskette drive grinds; system displays "Spelling check complete. Words marked: 0." Prompt replaced by "Type ID letter to choose ITEM; press ENTER:"	s5: spelling check complete	

*Figure 4.* The Menus Used in the Check Spelling Task on the IBM DisplayWriter. The Top Panel is the Task Selection Menu. The Bottom Panel is the Spell Tasks Menu.

---

<b>TASK SELECTION</b>	
<b><u>ID</u></b>	<b><u>ITEM</u></b>
a	Typing Tasks: Write, Revise, or Paginate Documents
b	Work Diskette Tasks: Delete or Duplicate Documents, Duplicate, Condense or Erase/Initialize (Name) Diskette Print Index of Diskette Contents, Change Document or Diskette Name, Recover Documents
c	Program Diskette Tasks: Default Formats, Duplicate Setups, Printer and Work Station Description. Duplicate and Erase Program Diskette
d	Spelling Tasks
Type ID letter to choose ITEM; press ENTER:*	

---

<b>SPELLING TASKS</b>	
<b><u>ID</u></b>	<b><u>ITEM</u></b>
a	Check Documents
b	Load Supplements
c	Clear Supplements
d	Store Supplement on Program Diskette
Type ID letter to choose ITEM; press ENTER:*	

---



heuristic places a link to \*spelling-tasks in the response from the action "choose \*spelling-tasks". This link not only indicates that the appearance of this heading is attributed to this action, but that there is an identity relation between them. In generalizing the analysis this identity relation would permit the specific item \*spelling-tasks to be replaced by any other. That is, the link placed by the Identity heuristic represents the conjecture that "choose \*typing-tasks" (for example) would produce a response that included \*typing-tasks.

The other aspects of the system's response to "choose \*spelling-tasks" do not participate in identity relations, so other heuristics must be applied to find attributions for them. In this case, since there are no earlier user actions to which they can be attributed they are all attributed to "choose \*spelling-tasks" by the default Previous Action heuristic, which attributes all unattributed consequences to the immediately previous user action. Since these links are not placed by the Identity heuristic they cannot be generalized: the analysis says nothing about what to choose if one wants something else to be shown in place of \*check-document, for example.

The complete analysis of the whole interaction is adequate to show how to obtain the system response "Spelling check complete", provided the task selection menu is on the screen initially. By tracing back the causal links, including prerequisite links, it is possible to reconstruct an adequate sequence of actions. But inspection of the latter parts of the interaction shows a serious problem in generalizing the interaction to check the spelling of some document other than the one in the example. Since the final system response, "spelling check complete", does not mention the document name or the diskette name, the EXPL analysis will not place identity links between the actions that type in these names and the system response. Therefore these actions cannot be generalized. If the feedback were changed to something like "Spelling check complete on document 'security' from diskette 'able' " then the Identity heuristic would work out how the names the user types influences what the system does.

Once a causal analysis of an example interaction is in hand it can be used to construct procedures to accomplish new but related goals. This generalization can be accomplished in more than one way; the implemented EXPL model can use either a planner-based method, in which new procedures are assembled from fragments extracted from analyzed examples, or an analogical generalizer based on Anderson and Thompson's PUPS system (1986). Lewis (1988) discusses other methods and the relationships among them. The use of causal analysis places EXPL in the category of *analysis-based* learning mechanisms, which contrasts with more familiar *inductive* methods. Inductive methods work by summarizing large numbers of examples in a way that suggests regularities, while analysis-based methods rely on insight into how one or a few examples work (see Lewis 1988, Lewis, Hair, and Schoenberg. 1989 for further discussion.)

### **3.2. Use of EXPL in Design**

The example shows that an EXPL analysis can point up issues in the design of feedback in a system. The interaction in the example is difficult to analyze because of the lack of repeated elements to connect the spell-check response to the information that produced it. Thus EXPL analysis provides complementary guidance to that available from GOMS or CCT analysis. It provides insight into the difficulty of specific aspects of an interaction but not into the overall effort required to learn to perform a task. However, there are three limitations of EXPL that mean that simply combining CCT and EXPL will not adequately support design for exploratory learning.

### **3.3. Limitations of EXPL**

First, EXPL analysis makes no use of goals. An actual learner observing the interaction in the example would probably connect the limited feedback about the completion of spelling correction to the goal of checking the spelling of some particular document, use this connection to produce a more meaningful encoding of the feedback, and in turn tie the entry of document and diskette names to this more complete representation of the feedback the

system provides. Thus, the learner would not miss the relationship between the document and diskette names and the actual outcome of the interaction. Indeed, knowing the goal of the interaction, the learner could guess the role of the document and diskette names at the time they were entered.

Second, EXPL analysis says virtually nothing about the adequacy of the prompts involved in the interaction. Some minimal adequacy of the prompts is implicitly assumed in the encoding of the menu choices and of the entry of names, in that the encoding must be able to relate the specific action (for example typing 'a') to its intent (specifying "spelling tasks"), and this relationship must be discovered by examining the prompts. However, EXPL analysis does not evaluate whether the prompts are adequate to permit someone to choose the appropriate action from among some available alternatives. For example, the menu from which 'a' for "spelling tasks" was selected might have also contained the items "correction task" and "spelling dictionaries". In this context the "spelling tasks" item description would probably not be adequate for learners to make the correct choice reliably, but EXPL, observing what choice was actually made in the example of the interaction it is analyzing, makes no attempt to assess this.

The third limitation of EXPL generalizes this last point: the model presupposes the availability of examples to learn from. However, in the case of an actual user trying to learn a new application, no examples are likely to be available, at least until the user gets far enough into the system to learn from the consequences of his or her actions. While EXPL provides an account of the process of making sense of the consequences of actions once they are chosen, it fails to address the decision processes by which these choices are made in an unfamiliar setting.

#### **4. PROBLEM-SOLVING IN UNFAMILIAR DOMAINS**

The missing piece in the model of learning by exploration is the process that actually produces the exploratory behavior, making choices in unfamiliar settings. The situation to be analyzed, in which the learner confronts a system offering an array of actions and chooses one, hoping to progress

toward some goal, is actually a familiar one in the psychological literature: classical puzzle problems have just this character. We can therefore attempt to bring over into the human-computer interaction domains models developed in the puzzle-problem literature.

In particular, we will import the notion of *problem space*, representing the structure of alternative actions available to the learner, and *search methods*, the processes by which the learner explores this space (Newell and Simon 1972, Newell 1980). We conjecture that what has been learned about search in puzzle tasks, in which solvers have little background knowledge to guide them, can be adapted to describe what learners will do in choosing actions in systems about which they have little knowledge.

#### 4.1. Problem Spaces

Learners who know only the goal and the available actions in a situation are operating in what Newell and Simon (1972) call the *basic problem space*. They lack such additional useful knowledge as ways to decompose a goal into subgoals, or how to effectively rank order available actions in terms of their contribution to accomplishing a goal. Having this added knowledge, or such other useful information as a partial description of the solution path or information about analogous tasks places the learner in an augmented *problem space*. A problem whose basic problem space is too large to search may be quite easy to solve in an augmented problem space. In the limit, there may be no search; a skilled user may simply know a *method* that specifies each correct step. However, new users will not possess such complete knowledge, and will have to engage in search.

#### 4.2. Search Methods

There are strong constraints on the search methods that are available to new users: there is good evidence that these methods are limited to variants of *means-ends analysis* (Polson and Jeffries, 1982; Greeno and Simon, 1988). Greeno and Simon (1988), in their review of the modern problem solving literature, conclude that the use of more powerful search methods by humans

depends on the use of augmented problem spaces, with knowledge of the structure of the space. Thus, the use of more powerful search methods depends on increasing expertise. There are two variants of means-ends analysis that are frequently observed in studies of novice problem solving: *hill-climbing* and *back-chaining*.

### **Hill Climbing**

Hill-climbing is a search strategy in which only currently legal actions are considered, and that action is selected that appears to offer the greatest progress towards the goal. In a typical puzzle problem it is possible to determine exactly what state an action will produce, so one can choose between actions by comparing the states they lead to to the goal. Solvers have a strong tendency to select actions that lead to states that *look like* the goal state, using perceptual similarity as a measure of distance. States that are on the true solution path but seem dissimilar to the goal are serious sources of difficulty. Solvers tend to reject actions that lead to such states and incur a big penalty in solution time (e.g., Atwood and Polson, 1976, Experiment 1; Jeffries, Polson, Razran, and Atwood, 1977).

### **Back Chaining**

Search using means-ends analysis in more complex tasks will attempt to select actions, not legal in the current state, that achieve the goal or appear to make progress toward the goal. Then the problem solver creates a subgoal to produce a state in which this action can be executed. Recursively attacking this subgoal leads to *back-chaining* or reasoning backwards. The planned solution sequence for the task develops backwards, beginning with the last action, the one that achieves the goal.

Larkin, McDermott, Simon, and Simon (1980) showed that freshman physics students used back-chaining to solve elementary mechanics problems. Since then, other investigators have shown that this strategy is used in a surprising number of different tasks by solvers with limited task domain knowledge.

Though back-chaining is widely observed, it requires knowledge which is not available to learners in the situations with which we must deal. The learner may know the repertoire of current actions, but in general does not know what actions may be available in the future, as required to do back-chaining. Further, even if the learner knew of the existence of a future action he or she would also need to know what the preconditions of that action are. For example, the action that accomplishes a goal may be offered only on one particular menu, and the novice does not know *a priori* what menu that is or how to get it.

Since back-chaining requires knowledge unlikely to be available, we conjecture that back-chaining plays a small role in the learner behavior we seek to model, and that hill-climbing will be the dominant search strategy learners will employ. However, the implementation of hill-climbing used for puzzle-problems cannot be applied directly by the computer learner: he or she cannot determine just what state will be produced by a particular action, so actions cannot be compared on the basis of the states they produce. A different approach must be used.

### 4.3. Label-Following

Recent studies in Polson's laboratory have shed light on how learners deal with this problem. Engelbeck (1986) identified a special case of the general hill-climbing strategy, which he called *label-following*. He observed that novice users have a strong tendency to select menu choices whose associated prompts share one or more terms with the description of their current task. Essentially learners determine which action appears to lead most quickly to the goal by comparing the *descriptions* of the available actions with a *description* of the goal.

This strategy works even if the new user has no clear understanding of the purpose of the task, e.g. 'condense diskette' on the IBM DisplayWriter. If a term appearing in the task description also appears in explanations of a choice

on several different menus, novice users tend to follow this path. Muncher (1989) also observed this strategy in subjects learning to use Lotus 1-2-3.

Notice that the label-following heuristic is closely related to EXPL's identity heuristic, despite the fact that EXPL is concerned only with the retrospective interpretation of actions, not the selection of future actions. The identity heuristic presumes that actions whose descriptions share terms with observed outcomes are likely to be causally connected to them. Label-following presumes that choices whose descriptions share terms with desired outcomes are likely to be causally connected to them.

Two sets of data show in detail the impact of the label-following strategy. The first is from Muncher (1989) where two groups of subjects had as their first task centering text in all cells of a spreadsheet. The second is from groups in studies by Polson, Muncher, and Engelbeck (1986) and Engelbeck (1986) who were learning the task of checking spelling using the IBM DisplayWriter, the task used in the GOMS, CCT, and EXPL examples above.

Muncher (1989) asked naive subjects to learn a series of eight tasks using Lotus 1-2-3. Subjects were permitted to make erroneous menu choices, but they were then stopped, and the experimenter undid the incorrect move and told them to select another move. Muncher only explicitly tutored subjects after several minutes of unsuccessful trial and error behavior. In most cases, there were a limited number of choices and subjects were capable of evaluating them exhaustively by trial and error.

Muncher constructed her task descriptions so that the simplest form of the label-following heuristic would not work for most menus. Thus subjects could not make menu choices based on a direct match between a component of a task description and a menu item. For example, one task description stated, "Center the text in all cells on the spreadsheet." The correct menu path required that subjects pick the selection WORKSHEET indicating that they wanted to manipulate the spreadsheet in the first menu. The correct choice at the second level was GLOBAL, indicating that they wanted to manipulate a global worksheet parameter. The third correct choice was

LABEL-PREFIX indicating that subjects wanted to manipulate the prefix character in front of any cell label that controlled its formatting (left justification, center, or right justification). The final menu had choices LEFT, CENTER or RIGHT, of which CENTER is correct. As can be seen, only the last of these choices can be made by label-following.

Two groups of subjects had this task as their first training task. Subjects had a very high error rate on the first three choices, and had to try many of the alternatives. In contrast, over 90% of the subjects made an initial correct choice on the final menu.

In the Polson et al. (1986) and Engelbeck (1986) experiments, subjects learned a series of tasks using a training procedure in which they were constrained to follow the correct path and were tutored after two attempts to make the correct response to a menu. Most tasks involved executing a series of menu selections and responding to prompts for task parameters like document name.

In both experiments, subjects were given the task of checking the spelling in a particular document. (See Figures 1-4 for details of the task.) This task occurred early in training sequences after subjects had learned how to make menu choices and enter parameters, but before they had experience with the spelling task or any related task on that path through the menu system. The components of the task description matched the relevant menu items, so that the first two menu choices can be made by label-following. Subjects made both correct choices with a very high probability.

These results suggest that learners' representations of a system are in large part dictated by the appearance and content of successive displays. Their interpretation of menu items and understanding of the consequences of selecting an item are limited to the conventional meanings of the words contained in each item. Novel uses of common words or technical terms in menu define actions with unknown consequences.



Background studies of puzzle solving, together with these observations, provide the basis for our account of learner's choice behavior with unfamiliar systems. We conjecture that this behavior can be modelled as heuristic search in a basic problem space defined by the available actions, and that at least when action descriptions are available, as on a menu, learners use the label-following variant of hill-climbing as their search strategy.

The problem-solving literature shows a wide variety of search strategies in use, varying with problem structure and solver knowledge. Our conjecture is that the learners we are attempting to model possess so little knowledge of the space they are searching, for the reasons given above, that this very simple model will provide a useful account of their behavior.

## 5. RELATING PROBLEM-SOLVING AND LEARNING

On the surface, an account of learners' problem-solving processes contributes little to a account of learning. But in fact modern cognitive theories of skill acquisition (ACT\*: Anderson, 1983; SOAR: Laird, Newell, and Rosenbloom, 1987) rely heavily on the interaction between general problem-solving and learning mechanisms. Problem-solving mechanisms attempt to achieve the learner's goal in a novel task environment. Learning mechanisms then encode the results of successful problem solving episodes as production rules that link the statement of a problem to its solution in a single processing step. If the need to solve the same problem occurs the solution is provided by applying the applicable production, rather than by repeating the earlier problem-solving episode.

Anderson (1987) assumes that the process of encoding productions is rapid so that discovery of correct actions by the general problem solver is the time- and resource-consuming aspect of learning. Thus, the goal of developing easily learned interfaces can only be achieved by designing interfaces so that they facilitate the problem-solving mechanisms learners employ. In this framework, this means facilitating the label-following strategy.

---

We note one complication, which brings out the importance of the feedback a system provides, as well as the prompts that support label-following. In general the outcome of an action in a computer system cannot be directly observed, in contrast with moves in puzzle problems. If feedback is poor, it may be impossible for the learner to determine whether progress is being made, or to determine just what the effect of the last action was. As a result, the productions cached by the learning mechanism could be wrong. In our proposed CE+ model we use the EXPL analysis methods to ascertain, after the fact, what the consequences of an action appear to have been.

## 6. THE CE+ MODEL

We can now describe how the elements discussed above can be combined in an integrated theory of exploratory learning of computer interfaces. We adopt the production representation of procedural knowledge from CCT, the analysis of the outcomes of actions from EXPL, the initial decision process from the puzzle-problem literature, and the coordination of problem-solving and learning from current cognitive architectures. We call the resulting theory CE+, to indicate both its origin in our earlier work on CCT and EXPL and the inclusion of significant additional ideas. In outline, CE+ consists of (1) a problem solving component, responsible for choosing actions; (2) a learning component, which analyzes the effects of choices and caches the results as CCT rules; and (3) an execution component capable of executing the resulting rules and coordinating execution of the rules with the problem-solving component. We discuss each component in turn.

### 6. 1. The Problem-solving Component of CE+

The problem-solving component of CE+ uses the label-following variant of hill-climbing that we described above. That is, faced with a choice among alternative actions CE+ chooses that action whose description overlaps most with its goal, provided that the action has not been tried before.

The details of the process by which this choice is made are taken from the Jeffries and Polson (1982) model of puzzle solving. The overall choice process is broken down into stages corresponding to the application of a set of heuristics which efficiently implement the overall decision. The first stage is a serial self-terminating search for an action whose description overlaps the goal more than a threshold amount. If a candidate action is found it is taken with a high probability if it has not been tried before, and with a lower probability if it has been tried before. If the first stage scans all actions without taking one, the second stage chooses an untried action at random, if any exist. If there are no untried actions the third stage applies one of two heuristics, depending on the number of actions. If there are three or fewer alternatives the degree of overlap between action and description and goal is determined for all alternatives, and the action with the most overlap is chosen. If there are more than three alternatives one is chosen at random.

As just mentioned, the details of this process are taken from a successful model of puzzle solving. We have made one adjustment: the original Jeffries and Polson model classifies actions as new or old on the basis of whether the *state* they produce has been visited before. Here, since the learner cannot determine what state an action will produce the choice process keeps track of whether or not the action itself has been chosen before in this context.

It may be that empirical work will reveal the need for further changes in the details of this model. For now we base our adoption of these particulars directly on our conjecture that the basic problem-solving processes are the same for learners confronting novel systems as for solvers confronting novel puzzles.

## 6.2. The Learning Component of CE+

Once the problem-solving component has selected an action, and it is carried out, the consequences of the action must be assessed and appropriate information stored as a guide to future action. To model users with very limited understanding of a system and task CE+ uses the same method used

in label-following to assess progress towards the goal: if the observed system response (assuming there is one) shares terms with the goal representation, the action is deemed to have been successful. If the action just taken is deemed unsuccessful, a subgoal is posted to undo it.

Whether the last action is deemed successful or unsuccessful, the learning component must store a representation of what happened, in a form that can be used to avoid having to perform problem-solving over again if a similar situation is encountered in future. This is done in two stages. First, a causal analysis, as in EXPL, is made of the last few user actions and system responses. This exposes the causal connections between actions and responses, and prerequisite relations among actions. It is then straightforward to build CCT productions that can repeat the appropriate actions when the observed outcome is desired in future.

The value of these productions is clear in success cases, but why bother to build them when the system response was not what was wanted? This can be seen as a form of incidental learning: even if the expected, and desired, result was not obtained it is still useful to know what happened and why. After all, the unwanted outcome now may be the wanted outcome in future.

Testing of the model will be needed to determine if this account of learning is adequate or needs modification. It is possible that people's evaluation of progress toward the goal is even cruder than that we propose. Mack, Lewis, and Carroll (1983) found subjects willing to accept almost any outcome of their actions as correct. It may be that productions built from incorrect actions should only rarely be stored. And it may be that information about the effects of choices should be stored declaratively as well as in production form, so as to be accessible to the selection process.

### **6.3. The Execution Component of CE+**

Once CE+ has productions available from earlier problem-solving episodes it needs a way to determine whether problem-solving must be invoked in a particular situation or whether existing productions can be used. CE+ uses the

simple criterion that if any production is applicable it should take precedence over new problem-solving. At a high level this makes CE+ resemble SOAR (Lair, et. al, 1986): problem-solving is a response to an impasse in the application of other knowledge.

#### **6.4. A Related Model**

Muncher (1989) developed a model of learning menu-based systems by exploration that is closely related to CE+. Her model is also derived from the GOMS/CCT and from the Polson and Jeffries (1982) models of the solution of puzzle-like problems. Muncher's model focuses on the problem solving component. She performed a transfer experiments that was originally designed to test an extension of CCT to tasks involving spreadsheets. She evaluated her model of the problem solving component by deriving predictions for the relative difficulties of learning of menu transitions required to perform series of tasks using Lotus 1-2-3. The results from the first task in two of her experimental conditions were discussed in Section 4.3.

Muncher classified new menu transitions as difficult, moderate, or easy depending upon whether they violated or were consistent with a hill-climbing search method that was similar to the method described in Section 6.1. Her data strongly supported her difficulty predictions. For example, menu transitions that required users to learn to make choices that were dissimilar to their goals were significantly difficult to learn.

#### **6.5. A Detailed Example**

Figure 5 shows a trace of the operation of CE+ on the first part of the spell-check interaction presented earlier. This is a hand simulation: while some parts of the model have been implemented in other contexts, including the decision process, the EXPL analysis, and the translation between the results of EXPL analysis and CCT productions, these parts have not been integrated. The task given to the learner is "Check the document named 'security' for spelling errors." We presume that the learner's encoding of this task identifies "check for spelling errors" as the operation to be performed and a

---

**Figure 5 . A Trace of the Operation of CE+ on the First Part of the Spelling Check Task.**

Events

Instructions provided:  
"Check the document named  
'security' for spelling errors."

TASK SELECTION  
menu (see top of  
Figure 3) on screen.

User types 'd'

SPELLING TASKS menu  
appears (bottom of Figure 3)

CE+ Activities

Extract "check for spelling errors" as operation,  
and "'document named 'security' " as thing to  
operate on.

Extract "Typing Tasks", "Work Diskette Tasks",  
"Program Diskette Tasks" and "Spelling Tasks"  
as descriptions of available actions.

Compare each of these to "check for spelling  
errors".

"Spelling tasks" overlaps, so choose it.

Action is encoded as "choose \*spelling-tasks"  
(see EXPL example in Figure 4.)

Note overlap between "Spelling tasks" heading  
in system response and goal, "check for spelling  
errors"; presume progress has been made.

EXPL prerequisite link is placed between action  
and presence of "Spelling Tasks" item on  
screen.

EXPL Identity heuristic links the heading to  
previous action.

Build CCT production from EXPL analysis;  
see Figure 6.

document named "security" as the object of that operation. We further presume that the learner viewing the initial Task Selection menu (shown in Figure 4) can extract the descriptions of the alternatives it affords, "Typing tasks", "Work diskette tasks", "Program diskette tasks", and "Spelling tasks", and can identify these as referring to actions rather than to objects.

The decision process will now scan this set of alternative actions, comparing each description to the action portion of the goal. The first three descriptions share nothing with the "check for spelling errors", while the last, "Spelling tasks", does overlap. With high probability this choice will be made.

The system response is to display a menu headed "Spelling tasks". Since this overlaps with the goal, CE+ will accept this as progress, and not post an undo goal.

CE+ now analyzes the relationship between the system's response and the recent actions it has selected, using the EXPL analysis methods. In this case the identity heuristic places a causal connection between the appearance of the "Spelling tasks" menu and the previous action, since the action shares the description "Spelling tasks". Since CE+ has performed only one action, any other aspects of the system response would also be linked to this same action, by the previous action heuristic. The EXPL analysis also notes a prerequisite connection between the action described by "Spelling tasks" and the presence of the Task Selection menu on the screen.

The results of this EXPL analysis can now be cached in the form of a CCT production that will carry out the first step of a method for accomplishing the "check for spelling errors" goal. The condition of this production checks for the presence of this top goal, and for the presence of its own step identifier (this is part of CCT's control structure; the production builder simply simply assigns arbitrary identifiers to steps.) The condition also checks for the prerequisite assigned by the EXPL analysis. The action of the production deletes its step identifier (more of the CCT control scheme), carries out the action of choosing "Spelling tasks", and sets the step identifier of the next

step. The production builder has to recover that the selection of "Spelling tasks" was actually done by typing "d".

The resulting production, shown in Figure 6, corresponds to the production ChkSpIllDoc.P1 in the CCT analysis in Figure 2, but there are some differences, beyond the use of meaningless step identifiers in the CE+ production. The EXPL analysis notes that the occurrence of the item "Spelling Tasks" is a prerequisite to choosing it, but not that the Task Selection menu is a prerequisite. The goal statements differ because the CE+ analysis cannot determine from this much of the interaction that the specific document name 'security' could be replaced. Finally, the production in Figure 2 is written to interface to a separately-defined method for making menu choices, with which it communicates by posting NOTES in working memory and which it invokes by posting the goal "GO-TO-NEXT-MENU", while the CE+-produced production simply invokes a primitive operator to type the letter "d".

---

**Figure 6. An Example Production Generated by CE+ Corresponding to the Production ChkSpIllDoc.P1 in the CCT Analysis in Shown in Figure 2.**

```
(SomeLabel
  IF ((GOAL CHECK-FOR-SPELLING-ERRORS-DOCUMENT-SECURITY)
      (STEP S333)
      (SCREEN SPELLING-TASKS))
  THEN
    ((Delete STEP S333)
     (PressKey d)
     (Add STEP S331)))
```

---

To use the resulting production a start-up production, corresponding to StartChkSpIllDoc in Figure 2, must be built to invoke it. The start-up production contains only control information and can be built for any method without reference to any information about how the method is performed, other than the identifier used for the first step. Once these



productions are built CE+ would not repeat its decision process in facing the Task Selection menu with the goal of checking spelling. Rather, these productions would apply and "d" would be typed immediately.

Recall that the EXPL analysis did not recover all of the structure of the spell-check interaction in Figure 3. Specifically, since the system response that signals the completion of the check does not mention the document name, the analysis did not determine the effect of typing in the document name. This problem can be avoided in the context of CE+, since the goal of the interaction is available during analysis of the results of actions. Since the name 'security' that the user types in is identical occurs in the goal statement, the production builder can determine that 'security' is not a constant, like 'd', but should be replaced by a reference to the goal. At the same time this identity relation permits CE+ to conjecture that the specific name 'security' could be replaced by some other in using the method in future.

## 7. DESIGN FOR SUCCESSFUL GUESSING

We have divided learning by exploration into a problem-solving phase, in which correct actions are discovered, and a learning phase, in which successful sequences of actions are cached as production rules. The problem-solving phase is the critical one, since the learning phase operates only on its output. The problem-solving mechanisms we have proposed are not robust, and seemingly minor flaws in the design of a user interface can completely derail them. Thus, we argue that the design of an interface that must be learnable by exploration must focus on facilitating the problem solving mechanisms.

Given the CE+ model, we can read off a set of design principles that will facilitate learner's problem-solving. Because the knowledge-poor strategy used in CE+ could fairly be described as a guessing process, we call the collected suggestions Design-for-Successful-Guessing. The principles are listed in Figure 7.

---

**Figure 7. Design Principles for Successful Guessing**

1. Make the repertoire of available actions salient.
2. Use identity cues between actions and user goals as much as possible.
3. Use identity cues between system responses and user goals as much as possible.
4. Provide an obvious way to undo actions.
5. Make available actions easy to discriminate.
6. Offer few alternatives.
7. Tolerate at most one hard to understand action in a repertoire.
8. Require as few choices as possible.

---

*Make the repertoire of available actions salient.* The CE+ model assumes a set of actions among which to choose. Traditional command languages cripple CE+ at the start, because there is no explicitly presented repertoire of actions. Menu systems which include some choices that are not visible on the menus are also bad. For example the IBM DisplayWriter, a menu-based system, used a keypress to summon a critical auxiliary menu, but this keypress was never listed as an action on any menu. Engelbeck (1986) found that operations involving this keypress were difficult to learn and recall.

*Use identity cues between actions and user goals as much as possible.* The label-following heuristic needs shared terms to work. Thus menu prompts, or command names, or other material that can be associated with actions, should share terms with user goal representations. The IBM Interactive Chart Utility required users to select colors by number, not name or by choosing

among color patches, even when executed on a color terminal. Users had trouble making appropriate choices even in simple situations.

*Use identity cues between system responses and user goals as much as possible.* Similarly, users must be able to gauge whether an action moved toward a goal or not. Identity cues support this. WYSIWIG editors provide continuing feedback about the effect of user actions that can be directly related to user goals. Of course systems which provide *no* system responses are especially bad. Many UNIX commands provide no response, which may be fine for the user who knows what they do but is hopeless for the learner.

*Provide an obvious way to undo actions.* If an action is seen to be a mistake users will have the goal of retracting it.

*Make available actions easy to discriminate.* The hill-climbing decision process in CE+ will work poorly if available actions seem to do similar things: it will be difficult to determine which action best approaches the goal. A choice among many ways to delete, say, which differ in the size of the unit deleted and whether it is replaced by blanks or not, is hard to make early in learning, as Mack, Lewis and Carroll (1983) report, because learners see these operations as very similar.

*Offer few alternatives.* Other things being equal, the chance of guessing the correct alternative goes down with the number of possibilities.

*Tolerate at most one hard to understand action in a repertoire.* The hill-climbing decision procedure can eventually make the right choice when all but one action can be seen to be inappropriate. But if there is more than one choice whose consequences cannot be projected the probability of error is high. Of course having *no* hard to understand choices is best, but designers of complex systems may not be able to ensure this.

*Require as few choices as possible.* Other things being equal, a longer series of choices has less chance of being completed successfully.

## 8. NORMAN'S ACTION MODEL AND DESIGN GUIDELINES.

Norman (1986, 1988; see also Hutchins, Hollan, and Norman 1986) applies a general theory of action to the problem of operating a computer system. In outline, the user's problem is seen as that of building two bridges, one across the 'Gulf of Execution', which separates the user's conception of a goal from the actions needed to accomplish it, and a second across the 'Gulf of Evaluation', which separates the results provided by the system from the user's understanding of progress. CE+ can be seen as a simple special case of this model, in which both gulfs are bridged (crudely) by determining the overlap of terms between goal statements, menu prompts, and system responses.

---

### *Figure 8.* Norman's Design Principles.

1. Use both knowledge in the world and knowledge in the head..
2. Simplify the structure of the task.
3. Make things visible: bridge the Gulfs of Execution and Evaluation.
4. Get the mapping right. Make sure that the user can determine the relationships between: 1) intentions and possible actions, 2) actions and their effects on the system, 3) the actual system state and what is perceivable by sight, sound, or feel, and 4) the perceived system state and the goals and intentions of the user.
5. Exploit the power of constraints, both natural and artificial.
6. Design for error.
7. When all else fails, standardize.

Norman (1988) derives design guidelines from this framework, summarized in Figure 8. We see considerable agreement between these principles and Design for Successful Guessing, but also some interesting points of difference. First, Design for Successful Guessing makes more specific suggestions than Norman's principles 3 and 4, while addressing the same issues. Principle 2, "Simplify the structure of the task", is reflected in our advice to keep the sequence of choices short, while Principle 5, on exploiting constraints, is a more general version of our advice to limit the available alternatives. Principle 6, "Design for error", shows up in CE+ only in the specific injunction to provide a means of backing up. In all these cases CE+ offers specificity, in suggesting how to design for learners using very crude, lowest-common-denominator problem-solving methods, but sacrifices generality. Clearly overlap of terms does not exhaust the possible cues for assessing progress towards a goal, for example, though it may be the most robust one.

Second, Norman's Principles 1 and 7 draw on facts about memory not modelled in CE+. The CE+ model does not include any process that could use declarative knowledge about one system to help in mastering a new one. Nor does it have difficulty retrieving the productions it stores, which are its only representation of earlier experience with a system. So the use of cues to assist in recall, and the value of standardization in reducing the requirements for new learning are lost in the CE+ framework, except when two situations can be made so similar that the same productions are applicable in each. This seems too restrictive, though the history of psychology is full of disappointed expectations of transfer in problem-solving situations (Gick and Holyoak, 1983).

Finally, Design for Successful Guessing provides for learners reasoning by elimination, if only as a last resort, while Norman's principles do not explicitly recognize this possibility. Support for this could be seen as falling within the general Principle 4, "Get the mapping right".

Overall, it appears that CE+ offers a conservative specialization of Norman's framework. It suggests specific ways to support the most basic problem-

solving methods, likely to be used by complete novices, while saying nothing about how to cater for more sophisticated learners.

## 9. DISCUSSION

We have argued that CE+ gives an account of exploratory learning of computer systems. We have derived specific design guidance from the framework that CE+ provides.

A gap remaining in the theory is the role of prior knowledge in the guessing process, as just noted in our discussion of Norman's design principles. CCT provides an approach to assessing transfer from these other systems, in terms of total learning required, but we have not worked out an account of the role (if any) of transfer in guessing.

Another issue remaining to be addressed is the role of user goals in design. Design decisions balance delicately on just what goals users will have, and how they represent them, something our present framework takes as given. We see two distinct problems here.

First, it is clear that users are capable of restating goals in ways that can profoundly affect the ease with which a task can be accomplished. For example, in a study of a messaging system learners typically transformed the stated goal "play all your messages and then delete them" into "for each of your messages, play it and delete it", which happened to be much easier to accomplish in the system we were studying. We do not know how to predict or control these goal transformations.

Second, while some user goals arise in the users' environment, and hence precede design, many others are part of users' responses to the design, and hence are influenced by the design itself. Our theoretical framework does not help to lay out what these "derived" user goals will be.

**Acknowledgments.** We have benefitted from extensive discussions with our students, Elizabeth Muncher, George Engelbeck, Kathleen Wharton, and Adrienne Lee, concerning many of the issues discussed in this paper. While we were writing this paper, Elizabeth Muncher completed her Masters Thesis in which she developed a model that has many similarities to CE+. Stuart Card, Jonathan Grudin, and an anonymous reviewer made many insightful comments which led to substantial clarification of many of the issues discussed in this paper.

**Support.** This research was supported by US West Advanced Technologies under their Sponsored Research Program. Dr. Catherine Marshall, our contract monitor, made important contributions to our thinking. The conclusions expressed are those of the authors any may not reflect the opinions of US West. Clayton Lewis acknowledges the support of the Office of Naval Research that led to the development of EXPL.

**REFERENCES**

- Anderson, J.R. (1983). *The Architecture of Cognition*. Cambridge, MA: Harvard University Press.
- Anderson, J.R. (1987) Skill acquisition: Compilation of weak-method solutions. *Psychological Review*, 94, 192-211.
- Anderson, J.R. and Thompson, R. (1986, June). *Use of analogy in a production system architecture*. Paper presented at the Illinois Workshop on Similarity and Analogy, Campaign-Urbana, Ill.
- Atwood, M.E. and Polson, P.G. (1976) A process model for water jug problems. *Cognitive Psychology*, 8, 191-216.
- Bennett, J., Lorch, D., Kieras, D. E., and Polson, P. G. (1987) Developing a user interface technology for use in industry. *Proceeding of Interact87, 2nd IFIP Conference on Human- Computer Interaction*, Stuttgart, September 1987.
- Bovair, S., Kieras, K. E., and Polson, P. G.(1988) The acquisition and performance of text editing skill: A production system analysis. Technical Report. University of Michigan (Submitted to *Human Computer Interaction*)
- Card, S.K. and Newell, A. (1985) The prospects for psychological science in human-computer interaction. *Human-Computer Interaction*, 1, 209-242.
- Card, S.K., Moran, T.P., and Newell, A. (1983). *The psychology of human-computer interaction*. Hillsdale, N.J.: Erlbaum.



- Carroll, J.M. and Rosson, M.B. (1987) The paradox of the active user. In J.M. Carroll (Ed.). *Interfacing thought: Cognitive aspects of human-computer interaction*. Cambridge, MA: Bradford Books/MIT Press.
- Engelbeck, G.E. (1986). Exceptions to generalizations: Implications for formal models of human-computer interaction. Unpublished Masters Thesis, Department of Psychology, University of Colorado, Boulder, CO.
- Gick, M.L. and Holyoak, K.J. (1983). Schema induction and analogical transfer. *Cognitive Psychology*, 12, 306-355.
- Greeno, J.G. and Simon, H.A. (1988) Problem Solving and Reasoning. In R.C. Atkinson, R. Herrnstein, G. Lindzey, and R.D. Luce (Eds.), *Steven's handbook of experimental psychology*. New York: John Wiley and Sons.
- Hutchins, E.L., Hollan, J.D., and Norman, D.A. (1985) Direct Manipulation Interfaces. *Human-Computer Interaction*, 1, 311-338.
- Jeffries, R., Polson, P.G., Razran, L., and Atwood, M.E. (1977) A process model for Missionaries-Cannibals and other river crossing problems. *Cognitive Psychology*, 9, 412-440.
- Karat, J., Boyes, L., Weisgerber, S., and Schafer, C. (1986). Transfer between word processing systems. In M. Mantei and P. Orbeton (Eds.), *Proceedings CHI'86 Human Factors in Computer Systems*. (pp. 67-71). New York: Association for Computing Machinery.
- Karat, J., Fowler, R., and Gravelle, M (1987). Evaluating user interface complexity: Experiences with a formal model. *Proceeding of Interact87, 2nd IFIP Conference on Human- Computer Interaction, Stuttgart, September 1987*.

- Kieras, D.E. (1988) Towards a Practical GOMS Model Methodology for User Interface Design. In M. Helander (Ed.) *The Handbook of Human-Computer Interaction*. Amsterdam, NV: North-Holland.
- Kieras, D.E. and Polson, P.G. (1985). An approach to the formal analysis of user complexity. *International Journal of Man-Machine Studies*, 22, 365-394.
- Kintsch, W. (1974). *The representation of meaning in memory*. Hillsdale, N.J.: Lawrence Erlbaum Associates.
- Laird, J., Newell, A., and Rosenbloom, P. (1987) SOAR: An architecture for general Intelligence. *Artificial Intelligence*, 33, 1-64.
- Larkin, J. H., McDermott, D., Simon, D.P., and Simon, H. A. (1980). Expert and novice performance in solving physics problems. *Science*, 208, 1335-1342.
- Lewis, C.H. (1986). A model of mental model construction. In *Proceedings CHI'86 Human Factors in Computer Systems*. New York: Association for Computing Machinery
- Lewis, C.H. (1988) How and why to learn why: Analysis-based generalization of procedures. *Cognitive Science*, 12, 211-256.
- Lewis, C.H. Hair, D.C., and Schoenberg, V. (1989) Generalization, consistency, and control. In K. Bice and C. Lewis (Eds.), *Proceedings CHI'89 Human Factors in Computer Systems*. (pp. 1-5). New York: Association for Computing Machinery.
- Lewis, C.H., Casner, S, Schoenberg, V and Blake, M. (1987) Analysis-based learning in human-computer interaction. *Proceeding of Interact87, 2nd IFIP Conference on Human- Computer Interaction*, Stuttgart, September 1987.

- Mack, R., Lewis, C. and Carroll, J (1983) Learning to use word processors: Problems and prospects. *ACM Transactions of Office Information Systems*, 1, 254-271.
- Muncher, L. (1988) The Acquisition of Spreadsheet Skills. Unpublished Masters Thesis. Department of Psychology, University of Colorado, Boulder, CO.
- Newell, A. (1980). Reasoning, problem solving, and decision processes: The problem space as a fundamental category. In R. Nickerson (Ed.) *Attention and Performance VIII*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Newell, A., and Simon, H.A. (1972) *Human problem solving*. Englewood Cliffs, New Jersey: Prentice-Hall.
- Norman, D.A. (1986) Cognitive Engineering. In D.A. Norman and S.W. Draper (Eds.) *User Centered Systems Design: New perspectives in human-computer interaction*. Hillsdale, NJ: Lawrence Erlbaum Assoc.
- Norman, D.A. (1988) *The Psychology of Everyday Things*. New York, NY: Basic Books.
- Polson, P.G. (1987) A quantitative theory of human-computer interaction. In J.M. Carroll (Ed.). *Interfacing thought: Cognitive Aspects of Human-Computer Interaction*. Cambridge, MA: Bradford Books/MIT Press.
- Polson, P.G. (1988). Transfer and Retention. In R. Guindon (Ed.), *Cognitive science and its application for human-computer interaction*. Hillsdale, NJ: Erlbaum.
- Polson, P. G. and Jeffries, R. (1982) Problem solving as search and understanding. In R. J. Sternberg (Ed.) *Advances in the Psychology of*
-

*Human Intelligence*, Vol. I. N.J.:Lawrence Erlbaum Associates, pp. 367-412

Polson, P.G., Muncher, E., and Engelbeck, G. (1986). A test of a common elements theory of transfer. In M. Mantei and P. Orbeton (Eds.), *Proceedings CHI'86 Human factors in computer systems* (pp. 78-83). New York: Association for Computing Machinery.

Rubenstein, R. and Hersh, H.M. (1984) *The human factor: Designing computer systems for people*. Burlington,MA: Digital Press.

Smith, S.L. and Mosier, J.N. (1986) Guidelines for designing the user interface software. Bedford, MS: Mitre Corporation. Report 7 MTR-10090, Esd-Tr-86-278

Thorndike, E.L. and Woodward, R.S. (1901) The influence of improvement in one mental function upon the efficiency of other functions. *Psychological Review*, 8, 247-261.