LEARNING STRATEGIES AND EXPLORATORY BEHAVIOR OF INTERACTIVE COMPUTER USERS

John Franklin Rieman

CU-CS-723-94

University of Colorado at Boulder
DEPARTMENT OF COMPUTER SCIENCE

LEARNING STRATEGIES AND EXPLORATORY BEHAVIOR OF INTERACTIVE COMPUTER USERS

by
JOHN FRANKLIN RIEMAN
B.A., Metropolitan State College, 1975
J.D., University of Colorado School of Law, 1978
M.S., University of Colorado, 1990

A dissertation submitted to the
Faculty of the Graduate School of the
University of Colorado in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science

1994

Abstract

Users may engage in "exploratory learning" to investigate the capabilities of a new interface. In this form of behavior, the user's short-term goals are not clearly defined, even if there is a general long-term goal to learn the software. Related exploratory behavior may occur when real tasks are involved that are not well specified, or within a system that is not entirely understood. It has been suggested that exploratory learning is effective and attractive to users, but there has been little investigation of its occurrence outside of laboratory and training situations.

The research described in this dissertation begins by examining the task of exploration in an abstract sense. The difficulties revealed by this formal approach help to guide a field study into the behavior and attitudes of computer users in everyday working situations. To provide further detail on the behavior found in the field studies, several cognitive models are implemented that describe how the user's general task description and the computer display interact to form interface-specific goals. Empirical work in the laboratory investigates the predictions of the formal theory and the model.

The formal theory shows that truly task-free exploration is exceptionally difficult. The majority of users studied in the field were found to avoid this kind of behavior, preferring task-oriented exploration supported by manuals and personal interactions. The cognitive modeling and laboratory studies revealed that the strategies effective for task-oriented exploration included label-following supplemented by active search for appropriate labels, along with a dual-search strategy that delves slowly deeper into both the actual interface and the users' associative memory.

Acknowledgments

This research is the result of several years of productive and rewarding collaboration with Clayton Lewis and Peter Polson. It also reflects a strong belief that human-computer interaction must be studied as part of a larger situation, a view that Gerhard Fischer and his research group have helped me to appreciate.

My thoughts on exploration and on how to "do science" have been further shaped by discussions with Jim Martin, Walter Kintsch, Susanna Cumming and my co-workers in the computer science and psychology departments. Marita Franzke, David Redmiles, and Evelyn Ferstl deserve special thanks for their friendship and advice.

Fourteen anonymous informants volunteered their time for a week of diary logging, followed by an in-depth interview covering their activities and attitudes toward exploration. This research could not have been done without their cooperation, and it is greatly appreciated.

This work has been supported by grant number IRI-9116640 from the National Science Foundation and by grants from U S West Advanced Technologies.

Contents

Figures

Tables

Chapter 1

EXPLORATION AND ITS IMPORTANCE

In this chapter, exploration is defined as an information-gathering activity with ill-defined goals. Exploration of computer systems by users is described. This is shown to be important for both learning and using computers. Research directions towards designing explorable software are introduced.

The research described in this dissertation concerns users' efforts to "explore" computer systems. Exploration is a widely used term: the Phoenicians explored the ancient seas, geologists explore the desert for oil reserves, the Voyager spacecraft explored the solar system, and a host of books and articles have titles such as "Rilke in Transition: an Exploration of His Earliest Poetry" (Rolleston, 1970). These various uses suggest the elements of a prototype definition for exploration:

- It is an information-gathering activity.
- It involves movement through an environment, physical or symbolic.
- It is similar to search, but with ill-defined goals.

It is the lack of well-defined goals that makes exploration especially intriguing, both from an algorithmic and a cognitive standpoint. A search algorithm is terminated when its goal is achieved, and in some applications search can be made more efficient by homing in on the goal (Korf, 1988; Winston, 1984). But with no clear goals, when should exploration terminate, and how can it be guided? Similar difficulties apply when we try to fit exploration into the search-like paradigm of means-ends problem solving, the weak method likely to be applied by human problem-solvers who lack domain-specific knowledge (Newell and Simon, 1972; Newell, 1990). If the end is no clearly specified, how can the means be chosen to approach it? These issues, in the specific domain of human-computer interaction, are the central focus of this research.

1.1 Exploration by Computer Users

What kinds of exploration occur as humans interact with computer systems? What are the external characteristics of these activities, and how do they relate to cognition and knowledge? There are no established answers to these questions, but we can propose categories that will help to illuminate the area at which this research is directed.

1.1.1 Categories of Exploratory Behavior

Three general categories of exploratory behavior can be distinguished. The first is *discovery-oriented exploration,* the activity that might takes place when a curious user logs onto a new system for the first time, or when an employee receives a new software package but has no immediate need for it. In these situations, some users may explore the new system with no clear goal in mind. They may look at menus, try out some of the software's functions, perhaps set options or defaults for future use. This behavior could be very open-ended: easily drawn to curious features, easily frustrated by difficult command sequences, subject to sudden termination by external pressures or the resistance of the interface.

Task-oriented exploration, on the other hand, is more constrained. This behavior occurs when a user is trying to accomplish a general task but doesn't know what features the software might offer towards a solution. For example, a user who has only used word processing for correspondence may be assigned the job of producing a company newsletter. How will this user discover features such as multiple-column text, special fonts for headlines, automatic pagination, and the like? Some users may look for examples supplied with the software, but others may poke about in the interface, discovering potentially useful features and trying them out. Although constrained by the task, task-oriented exploration is still more open-ended than searching for a specific command, even one that the user isn't sure exists. When exploring, there could always be another hidden feature, another combination of options that would yield an even better solution to the current task.

A third type of behavior, *problem-oriented exploration,* is still more focused. This is the activity that takes place when a user has a well-defined subgoal within a task-oriented situation, but has no clear idea as to how the subgoal can be accomplished. The user attempts to solve the problem by searching through the interface, trying different controls, and observing their effects. This behavior could more accurately be termed "search," but it has generally been referred to as "exploratory learning" (Polson & Lewis, 1990).

The distinctions made between the three kinds of exploration are not sharply defined. They represent points within a range of behavior, a range that can be extended further to include the behavior of a skilled user with a clearly defined task. The dimension on which these behaviors differ is related to the user's "goals" as the behavior progresses.

1.1.2 Goals and Situated Actions in Exploration

The term "goal" is used here to describe the results or effects that the user attempts to achieve with an interface. In studying human-computer interaction, the distinction has been made between "task goals" and "device goals" (Payne, Squibb, & Howes, 1990) Task goals arise independently of the computer system. A word-processing task goal, for example, might be to make the title of a report stand out from the rest of the text. Device goals arise from the computer system and it's interface. A clear example of a task goal would be: "select Bold from the Format menu."

But there is no sharp distinction between task and device goals, and most of a user's intentions involve elements of both. Changing a report's title to boldface, for example, is a task-oriented goal that also includes the device-oriented element of "boldface." Because of this indistinction between the task and device goals, it may be more productive to consider a related but less abstract distinction. This distinction identifies the information source that leads to a goal's existence. In interacting with a computer, that source may be either the user's background knowledge or the immediate interactions with the interface. For most goals, the source will be some combination of these.

Applying this distinction yields the relationship shown in Figure 1.1. Note in the figure that the definitions of exploratory behavior and the distinctions between task and device goals are both represented as ranges. Discovery-oriented exploration falls at one end of the range, with essentially none of the user's goals defined by background knowledge when the exploratory session begins; these goals must arise out of the interface. In the middle of the range, both the user's knowledge and the interface help to define the goals in task-oriented exploration. The user may know,

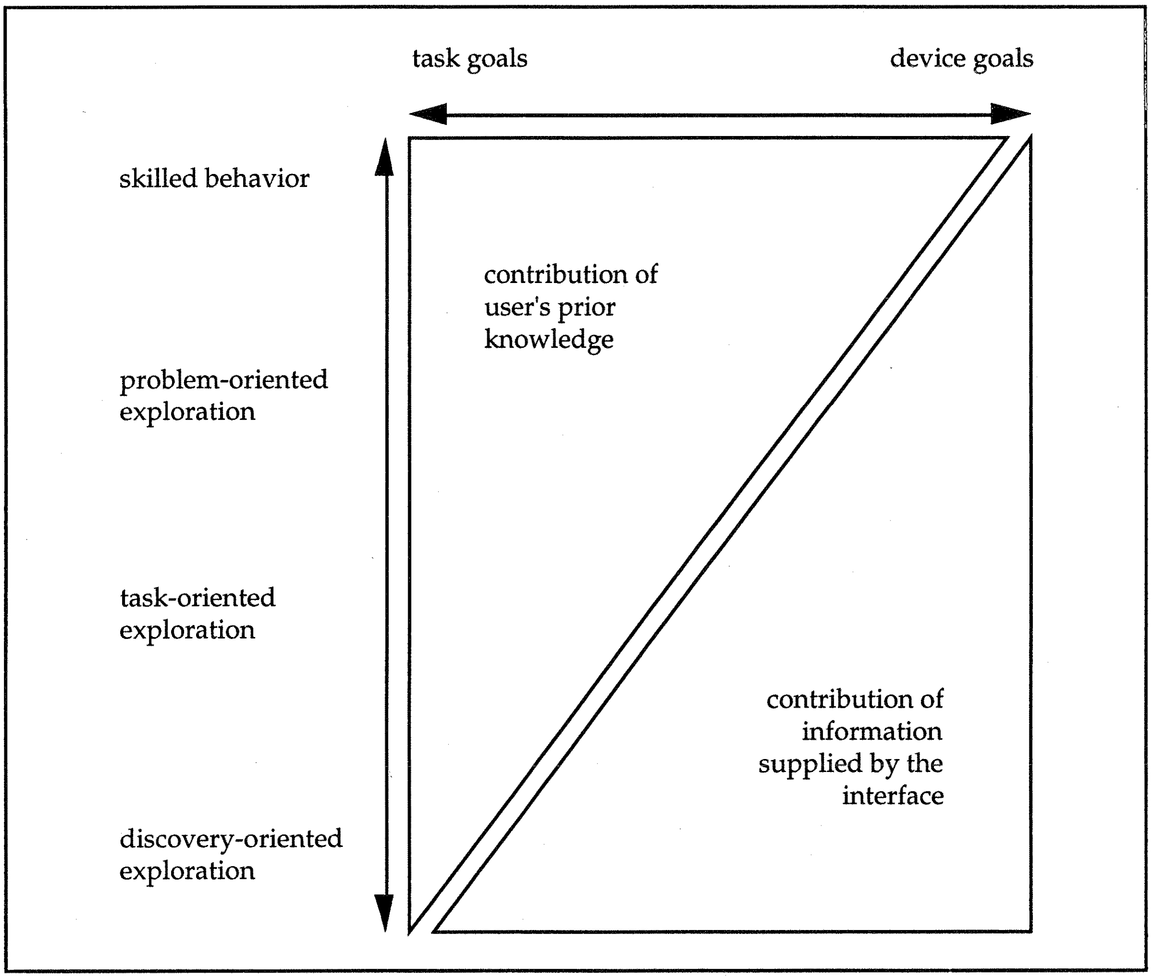

task goals
device goals
skilled behavior
problem-oriented exploration
task-oriented exploration
discovery-oriented exploration
contribution of user's prior knowledge
contribution of information supplied by the interface

Figure 1.1 Sources of task and device goals in forms of exploratory behavior.

for example, that a task requires a meaningful display of a set of data, but may have no idea of what display options a graphing package offers. At the other end of the range, with problem-oriented exploration, relatively low-level goals are defined by the task (i.e., "change the text to Symbol to see what that looks like"), leaving only the lowest-level subgoals to be defined by the interface (i.e, "select Symbol from the Font menu"). Extending this range to its obvious endpoint in the direction of user knowledge yields a description of highly skilled behavior, where even very low-level, interface-specific goals can be defined almost entirely without consulting the current state of the interface. At the other end of the scale, trivially, would be random behavior.

1.2 The Importance of Exploration

Why is exploration important? Can't people learn how to use software by reading the manuals or attending classes, and then simply rely "skilled behavior" to accomplish their tasks? There is evidence that these traditional approaches have failed for current systems, and there is reason to believe that they will continue to be unworkable.

Fischer (1987) has described the overwhelming complexity of modern workstation-based systems like UNIX, with more than 700 commands, and the Symbolics, with tens of thousands of functions. Even the IBM PC and the Macintosh personal computers, designed for use by nonspecialists, offer thousands of different software applications, many of them rivalling the complexity of applications designed for dedicated users in the UNIX environment. Fischer shows that users can't work with this software effectively because they don't know how to use many of the applications' functions. They may not even know that the functions exist.

Rosson (1984), Draper (1985), Nielson, Mack, Bergendorff, and Grischkowsky (1986) have all shown that knowledgeable, experienced users of complex systems like X-Edit (Rosson) and UNIX (Draper), and integrated business software packages (Nielson, et al.) master only an idiosyncratic subset of the total functionality of a system. The breadth of an individual's expertise does not seem to be related to either years of experience or technical background.

These studies, combined with various popular reports of under-utilization of information processing technology in all aspects of American life, support the contention that failure to use the full functionality of complex software is a major barrier to improved productivity. Experienced users make suboptimal use of tools that they use every day.

1.2.1 Exploration as an Alternative to Training

Training is not the solution to this problem. With dozens of software packages installed on a typical workstation, and hundreds of separate functions and options in each package, formal training can cover no more than core functionality. Many users consider even this basic instruction to be an intrusion on their daily workflow. Studies reviewed by Carroll (1990) and Carroll and Rosson (1987) conclude that people prefer, in fact almost insist on, learning software by exploration. New users are not interested in learning how to use an application; they want to get their work done. They explore the interface in an effort to accomplish their normal activities, learning bits and pieces of the system's functionality from day to day.

Dramatic evidence of this behavior is evident in the Mac lab in CU's Norlin library. The lab has more than 100 Macintoshes, available to students without any special training requirements. At any given time, the majority of those machines are running Microsoft Word, yet the lab has only a single copy of the Word manual set. Clearly, exploratory learning is common practice for most of these users.

1.2.2 Exploration as an Alternative to Knowledge

Underlying the discussion so far is an implicit assumption that users must know how to use a system before they can be productive. While this may be a workable approach for persons whose computer needs are well-defined, such as word-processing pool secretaries or operators of computer-controlled manufacturing systems, it is an approach that fails for users whose needs are widely varied. These users can't afford the time to learn functions that they may only use once before the software becomes obsolete. Users in rapidly changing software environments, such as a university network of UNIX workstations or personal computers, are especially affected by this dilemma.

For these users, task-oriented and problem-oriented exploration can have value even when little or no knowledge is retained. If a problem can easily be solved by exploration whenever it arises, then there is no need to learn the solution path. In non-computer explorations this occurs frequently: While attending a conference in another state, I find myself with a free evening in a university town. I wander around the shops near the edge of campus, perhaps finding a used book store to browse, or a foreign film theater, or a club with local musicians. I don't need to learn the addresses of these places, any more than I need to know the addresses of similar places in Boulder. I can always find them, or new shops, by exploration.

Similarly, the experienced Macintosh or PC user may download games and other applications from an electronic bulletin board or anonymous ftp site, try out the applications briefly, and discard most of them. The experienced UNIX user on a networked system will make similar forays into the site's ever-growing collection of software tools. Learning is not a major issue here; discovering important features is.

1.2.3 Designing Software for Effective Exploration

The difficulties of training and the preferences of users strongly suggest that the most effective way to help users access the full functionality of powerful computer systems is to design the software to facilitate untrained use. With little analysis we can predict that explorable software must make the existence of relevant functionality obvious to the user, and it must support the cognitive processes involved in exploratory problem solving and learning.

But applying these general guidelines to the design of explorable systems is hard. Even designers with good intentions can produce systems, like the Apple Lisa, that are essentially unusable without some training (Carroll and Mazur, 1986). Advanced features in complex software seems especially hard to discover through exploration. How many novice users of Microsoft Word 4.0 can effectively use Word's powerful style sheets? How many even know the feature exists, hidden as it is behind the cryptic menu item "Full Menus"?

Software designers' intuition seems to fall short when it comes to producing explorable systems. The research described in this dissertation was designed to provide a firmer foundation for those efforts.

1.3 Results of Previous Research

Software can best be designed to facilitate exploration if we have a clear picture of low-level exploratory behaviors and of the context in which they occur. How do users structure their explorations of computer systems? In what situations do they choose to engage in exploration, and when do they abandon that approach as ineffective? When they do explore, do they follow well-defined algorithms, similar to those defined in artificial intelligence research for search activities? Do they just thrash about, noticing items at random and missing major portions of the interface? Do their methods for computer exploration vary significantly with their knowledge of interface and system conventions, or is there some general approach to exploration that novice and advanced users apply alike?

For the class of users specifically targeted by this research, experienced users who are investigating new or unknown functionality within their "home" systems, the fundamental answer to these questions is that we simply don't know. But previous work hints at the behavior that may be found, while providing theoretical mechanisms to analyze that behavior.

1.3.1 Research into Exploration in a Broader Context

Interest in exploratory behavior predates the advent of interactive computer systems by at least a quarter of a century. In the 1950s, Berlyne (1960) and other researchers (review in Fowler, 1965) pursued a long and varied series of investigations into exploration and curiosity. The fundamental question driving this research was not *how* people explore but *why*. The issue arose in the context of then-current theories that required all behavior to be motivated by fundamental "drives," such as hunger, sex, and the need for sleep. Drive theories were hard pressed to explain curiosity and its attendant exploratory behavior, and Berlyn proposed to fill the theoretical breech.

Berlyne's ultimate solution was to propose additional drives. What he termed *specific exploration* reflected a basic need to reconcile conflicting data in the environment, while *diversive exploration* reflected a need to maintain the brain's information-processing system at an optimal level of activity (Berlyne, 1960). The two terms map roughly onto the distinction made earlier between task-oriented and discovery-oriented exploration.

The laboratory studies of the drive-oriented research program measured such things as time spent examining graphics of varying complexity. They didn't investigate the exploratory activities of humans in a complex environment. But they did discover that an explorer's attention to an item is greatest when the item is neither entirely familiar nor entirely inscrutable (Berlyne, 1960; Day, 1981), a finding that may describe a proximal rule for choosing among multiple paths when only some can be explored. It is important to recognize that this behavior depends on the explorer's existing knowledge of the situation.

More recent research has examined goal-free exploration for its own sake, rather than in reaction to a theoretical paradox. Most of this work is in the context of child development (see Gibson, 1988, for review) and animal behavior (reviews in Archer and Birke, 1983; Renner, 1991). It is easy to place children or animals in a novel situation and observe their behavior in the absence of well-specified goals, and it is clear that such exploratory behavior has value. As Voss (1987) points out, exploration exposes the causal relationships that the organism must know for future means-ends problem solving.

Similar studies of adult exploratory behavior are rare, in large part because it is difficult to observe spontaneous adult behavior in a well-controlled laboratory situation (Wohlwill, 1987). Some observational data has been collected in public areas such as museums (Görlitz, 1987; Koran, Morrison, Lehman, Koran, and Gandara, 1984), but for studies of adult exploration of complex systems we must look to the literature in computer-human interaction itself.

1.3.2 Research into Exploration of Computer Interfaces

Not long after interactive computer systems became widely available, several researchers recognized the success of computer games and argued that explorable user interfaces should have many of the same properties (Malone, 1982; Carroll, 1982; Shneiderman, 1983). Mastering the software should be intrinsically motivating, features should be revealed incrementally, and the system should be at least minimally useful with no formal training. Several research efforts reflecting this argument are described in the following subsections.

1.3.2.1 The minimalist instruction approach

For the purposes of this research, valuable insights can be gained from a series of studies begun by Carroll in 1981. This work initially analyzed the learnability of office-system interfaces in situations where novice users were given no coaching or training (Carroll, Mack, Lewis, Grischkowsky, and Robertson, 1985). The research demonstrated that most users are willing and able to learn by exploration, although there are individual differences in exploratory aggressiveness (Carroll, 1990; see also Neal, 1987). It also showed that novices attempting to explore a totally new system often make major and irrecoverable errors, even with the aid of basic manuals and tutorials (Carroll and Mazur, 1986).

Carroll and his associates have continued this line of investigation, focusing on a "minimalist approach" to training and manuals (Carroll, 1990). The approach shows dramatic success in bringing new users up to speed in complex environments, and many users express a preference for this kind of guided exploratory learning.

Even with well-designed manuals, however, the success of discovery-oriented exploration can vary widely. Working with novice users in the domain of spreadsheets, Charney, Reder, and Kusbit (1990) compared the effectiveness of three methods for learning a subset of the program's commands: unguided exploration, untutored solving of experimenter-defined problems, and explicit guidance through the solutions to experimenter-defined problems. A simple manual was available to all the groups.

The subjects who who solved defined problems without tutoring performed best on later tests. Charney et al. conjecture that the subjects in the exploratory situation were able to create only a narrow range of goals, which failed to attract their attention to many of the features of the interface. This provides our first hint at a solution to the explorer's dilemma of how to structure exploration. Users who focus their explorations on appropriate "microtasks" may achieve greater success at discovering useful features of an interface.

1.3.2.2 Instructionless learning

But how do users behave when no manuals at all are available? Shrager (1985; Shrager and Klahr, 1986) investigated this question by giving undergraduates the unstructured task of learning to operate a "BigTrak" toy. The toy is a six-wheeled truck with a built-in computer controller, programmed with a simple keypad. Shrager reported that subjects began their exploration with an orientation phase, then engaged in a series of problem-solving episodes, in which hypotheses about keypad functions were generated and tested.

Shrager's computer model of the behavior demonstrated that much of the knowledge afforded by exploratory activities was ignored, with subjects noticing primarily the information that would confirm (not deny) their hypotheses. A cursory analysis suggests that the information-management difficulties of problem solving are great even within a simple interface. Even greater difficulties should arise for users of the complex interfaces available to most interactive users today.

Close examination of the two protocols reproduced in full in Shrager's (1985) dissertation reveals behavior that recalls the findings of Charney et al. The more successful of the two subjects frequently expresses his experiments as tasks to be achieved with the truck. After establishing that the ">" button plus a numerical argument causes BigTrak to turn, the subject says, "Let's see if I can get it to turn ninety degrees or whatever." He then tries numerical arguments (with hypotheses as to their meaning: degrees or minutes) until a 90-degree turn is achieved. The protocol from Shrager's less successful subject, who has less computer experience, shows fewer of these goal-oriented tasks and more examples of essentially random key presses followed by observation. In addition to the single-goal tasks, Shrager also observed subjects creating multistep tasks that exercised their growing understanding.

Do experienced interface explorers regularly create "microtasks" of varying complexity to provide structure for their exploration? How do they decide what to include in these tasks? How do the microtask goals interact with exploration and learning? These are some of the questions addressed in this research.

1.3.2.3 Label-following with simple interfaces

Ongoing work by Lewis and Polson has also investigated uninstructed behavior where no manuals are available. This work has examined the actions of users faced with well-defined tasks and novel interfaces. Engelbeck (1986) and Muncher (1989) have shown that the behavior of subjects faced with novel menus can be described by a label-following strategy, predicted by the identity heuristic in Lewis's (1988) EXPL theory. This research has also investigated the behavior of users with novel telephone answering-machine interfaces (Lewis, et al., 1990), and with a library database (Rieman et al., 1991).

But as a basis for understanding the exploratory behavior of users in their daily routine, all of these studies have a fundamental shortcoming: they focus almost exclusively on novice users of entirely novel systems. For most users, most of the time, the problem is to understand a new program or previously unused parts of an existing program in a well-understood environment, such as the Macintosh or Windows on a PC. The empirical and theoretical investigations described in this dissertation go beyond current work in exactly that direction.

1.4 The Value of a Multi-Threaded Research Approach

In the field of human-computer interaction (HCI), methods for studying user behavior can be informally divided into two approaches: experimental psychology in the laboratory and observations of users in the workplace. Both of these approaches are supported by theoretical foundations, including both traditional theory and theory embodied in cognitive modelling work.

The laboratory-based approach of experimental psychology grows out of a long history of work in the human factors field. In this style of research, investigators bring users into a laboratory situation and make quantitative observations of their behavior. The users are chosen from a single population, and the laboratory situation is carefully designed to vary only a few independent variables. The situation may have little in common with real computer usage. In the conservative version of this approach, a large number of subjects are tested, yielding data that can be interpreted with statistical methods and projected to a larger population (Landauer, 1988). A related method uses protocols taken from a few subjects and analyzed in depth, as in thinking-aloud studies or individual user modelling efforts (Lewis, 1982; Shrager & Klahr, 1986).

The second dominant approach is to collect data in the workplace, with the researcher often interacting closely with users in an informal way. The most radical version of this approach is participatory design, often identified with the European and particularly the Scandinavian HCI community (Floyd, Wolf-Michael, Reisen, Schmidt, & Wolf, 1989). In participatory design, users and designers work together to identify system goals and to iteratively produce systems that meet the users' evolving needs. Some other workplace-oriented methodologies include user surveys, monitoring the use of prototypes, and contextual design (Wixon, Holtzblatt, & Knox, 1990). Although it is not always the case, investigations in the workplace are generally less controlled than laboratory efforts, involving fewer subjects and relying more heavily on subjective judgments and personal interactions. These efforts are most often described as design methodologies, not research tools, and the data they yield tend to be less predictive, more anecdotal, more limited to the individual situation under investigation.

Most HCI researchers are aware of this distinction, and many design projects use a mixture of the two approaches (Gould, Boies, & Lewis, 1991). Nonetheless, the polarity is informative. On the one hand, the laboratory approach typically yields objective data that can be interpreted statistically to predict the behavior of a population, but the data concern the behavior of subjects in a contrived situation. Critics of this work, including proponents of the situated cognition view, argue that it has little impact on the usability of real systems (Suchman, 1987; Wixon, Holtzblatt, & Knox, 1990). By contrast, information gathered in the workplace as part of a design project reflects the real, day-to-day work of the participants, but extracting objective, general data from these reports may be difficult.

Insights into the interaction between these two general approaches can be gained by examining the interaction of similar approaches in another research field, animal behavior.

1.4.1 Two Approaches to the Study of Animal Behavior

The study of animal behavior is much older than HCI, with the roots of its modern theories going back at least as far as Pavlov and Darwin in the nineteenth century. But as with HCI, there are fundamentally two research approaches. The first of these is to study animals in the laboratory, an approach usually associated with experimental psychology. The second is to study animals in natural field situations, an approach taken by ethologists.

Like experimental psychology in HCI, experimental psychology with animals often involves contrived situations that are far from natural: rats are encouraged to visit the arms of radial mazes where cheese appears and disappears as if by magic; pigeons learn to dance from microswitch to microswitch in response to colored lights or photographs; cats are forced to escape from boxes with trick latching devices. Subject animals are carefully chosen from an artificially bred population, and groups for a given experimental condition may be surgically disabled, drugged, or raised in exceptional environments. But the data produced are clearly objective.

By contrast, ethological investigations are typically performed in the field, with randomly selected members of the naturally occurring population. Much of the work is simple visual observation of natural behavior. Other research relies on artificial devices, such as radio collars to track animal movement patterns, and some projects involve experimental manipulations, such as substitution of differently colored eggs in field sites. Ethologists strive to be objective, but the research is unavoidably influenced by behavioral coding decisions, and reported data are often based on small samples of a population. As with workplace HCI, some of this research may involve participation and highly subjective judgments: Konrad Lorenz with geese, Diane Fossey with gorillas. Interestingly, this approach, like workplace HCI, also has a history that is more closely associated with the European than the American research community.

1.4.2 Reconciling the Two Approaches

As with the two polar approaches to HCI research, there is clearly a tension between the two approaches to the study of animal behavior. However, several researchers have recently suggested that the two approaches can be complementary, in both a theoretical and a practical sense (Brain, 1988; Fantino & Abarca, 1985); Miller, 1985].

In a theoretical sense, the approaches can be seen to ask different, but complementary, questions. Ethological research in the field asks *functional* questions: how is the survivability of a species enhanced by a given behavioral characteristic, and why did that characteristic evolve? (Houston, Kacelnik, & McNamara,1982). To provide convincing answers to these questions necessarily requires that as much as possible of the organism's natural environment be considered in the analysis. On the other hand, experimental psychology in the laboratory asks *causal* (proximate) questions: what combination of cues does an organism perceive that causes it to behave in a certain way, and what cognitive algorithms does it use to define its behavior? The theoretical interplay is made especially interesting by the requirement that behavior which might be functionally optimal must be approximated by algorithms that an organism can process with its available memory and cognitive power (Fantino & Abarca, 1985).

It is this interaction between theoretical questions that also sets up the practical interaction between the two research approaches. The data and theories of ethologists suggest functionally optimal behavior; experimental psychologists can then design experiments to uncover mechanisms that implement or approximate this behavior.

1.4.3 Clark's Nutcracker: A Suite of Studies

As an example of the interaction between field and laboratory research in animal behavior, consider a series of studies investigating the behavior of Clark's Nutcracker, a bird found in the conifer forests in the western part of North America. Observations of naturalists have revealed that the seeds of pine trees form a major part of this species' diet. During the seed harvesting season, which lasts about one month, nutcrackers remove the seeds from pine cones and bury them in shallow holes, called caches, for retrieval during the rest of the year (Tombeck, 1978).

The relationship between the nutcracker and the pine trees is a close one. The beak of the nutcracker is specialized for removing seeds from cones and carrying them to cache sites or the nest, and the nutcracker's range closely parallels the range of the pine trees that provide its food. The range of the forest, in turn, is influenced by the nutcracker's activities: not all cached seeds are recovered, and unrecovered seeds may germinate and grow into new trees.

Field studies in the ethological style investigated this relationship between bird and forest, with particular attention to the nutcracker's effectiveness in caching and recovery (Tombeck, 1978; Vander Wall & Balda, 1977). Using a combination of energy calculations, field examinations of recovered caches, and direct observation, these researchers concluded that an individual bird must store on the order of 32,000 seeds each year, in roughly 8,000 caches. Although some of these caches are stolen by rodents, and others are unrecovered and left to germinate, an estimated 60 to 86 percent of the caches are located and recovered by nutcrackers.

These results of this investigation into functional issues raised causal questions that field research alone could not easily answer. Specifically, what were the mechanisms that nutcrackers used to achieve this high recovery rate? Was there some common pattern of storage and search to which all nutcrackers were privy, making the caches a communal resource? Or, did the birds actually possess the spatial memory needed to recall and recover their own caches? If so, did they rely on local, visual cues, such as the shape of a rocks and trees near the cache, or did they navigate using some more sophisticated system?

To answer these questions, researchers transferred the investigation from the field to the laboratory (Vander Wall, 1982; Kamil & Balda, 1985). A large outdoor flight cage was constructed, with a dirt floor and various objects, such as logs and rocks, to simulate a forest setting. In this setting, a series of controlled experiments eliminated olfactory and other possible cues that could not be controlled in the field. The results confirmed that a nutcracker's primary means of cache relocation was, indeed, spatial memory of the locations in which it had stored its own seeds, cued by large objects near to each cache.

1.4.4 Applying Similar Principles to HCI Research

The time-course of the nutcracker investigations illustrates an effective research paradigm. First, informal observations of naturalists identified the basic relationship between the nutcracker and the pine forest. Second, formal fieldwork investigated the functional relationships, raising questions as to the causal mechanisms involved. Third, laboratory experiments, designed to mimic important features of the natural environment, probed those mechanisms.

A similar complementary research approach can be effective in HCI research. Anecdotal evidence and insights acquired during participatory activities with users can suggest questions for more objective field studies in the workplace. These studies, in turn, can point to questions for investigation in the laboratory. They can also be used to suggest laboratory situations that reflect the critical constraints of the user's real environment. At each stage, these investigations can be tied together by formal theory and cognitive modelling.

1.5 Studying Exploration: an Overview of the Dissertation

The study of exploratory behavior described in this dissertation follows the general pattern just described. It begins with anecdotal observations, supported by some existing research, as described in this chapter. The questions raised by this discussion are further refined in theoretical work described in Chapters 2, which describes a formal model of the task of exploration. This formal work raises a number of questions concerning the extent and form of exploratory behavior, questions that can only be answered by objective fieldwork. That fieldwork, in the form of diary studies and associated interviews, is reported in Chapters 3 and 4.

The results of the fieldwork require a rethinking of the expectations raised in this chapter, and they define the direction for further, lower-level investigations of exploratory behavior. That investigation is performed in two arenas: a series of cognitive models of exploratory behavior are developed, as described in Chapter 5, and the predictions of those models and of the formal theory (Chapter 2) are tested by examining the behavior of users in a constrained laboratory situation, as described in Chapter 6. Finally, Chapter 7 summarizes the theoretical and empirical work and discusses what the results mean to interface designers and evaluators.

Chapter 2

A Formal Model of Exploration

A formal model of an exploration space and algorithms is described, with a mapping between the model and the interface to a computer system. Optimally explorable models and algorithms are developed. Information-management requirements are discussed. The model is shown to expose fundamental difficulties in exploration.

Some of the constraints and difficulties associated with exploration can be exposed by analyzing an abstract model of the exploration process. The two components of the model are a simplified description of the environment to be explored and a description of the activities of the explorer within that space. These components will be referred to as the *exploration space* and the *exploration algorithm.* The approach is similar to the formal analysis of search, in which the effectiveness of various search algorithms can be analyzed within search spaces of different sizes and configurations (Korf, 1988; Winston, 1984).

2.1 The Exploration Space

For the purposes of this research, the only situations analyzed are those offering a finite number of discrete actions that move the explorer among a finite set of discrete states. Thus, the analysis lays no claim to coverage of situations such as the geographical exploration of a featureless lake or plain. It is also assumed that all locations can be reached from the explorer's starting location.

For this discretized exploration, the space to be explored can be represented as directed graph. Each available exploratory move is represented by a directed arc, with the locations before and after the move represented by nodes, referred to as the source and target of the arc. Nodes and arcs are labelled. A node's label is accessible to the explorer only when the explorer is in that node. An arc's label is accessible only within

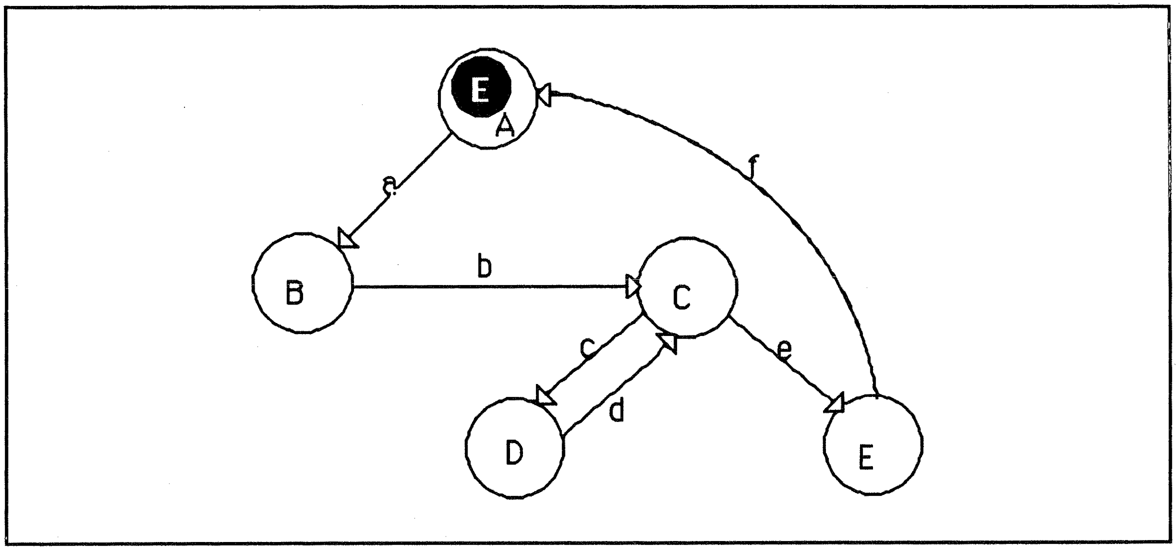


Figure 2.1 A simple exploration space.

the node that is the source of the arc. Figure 2.1 shows a simple space to be explored. A marker ("E") indicates the explorer's current position within the space.

The mapping between the model and an interactive computer system is this: Nodes represent states that the system can assume, and arcs represent commands that the user can invoke. The label on a node represents the perceptible characteristics of the state, while the label of the arc represents the command label. Further details of the mapping are discussed in Section 2.6.

2.2 Discovery-Oriented Exploration

We first consider what is necessary for discovery-oriented exploration. As defined in Chapter 1, this is the behavior that occurs when a user has no clear task goal, but endeavors to discover the functionality of the interface for its own sake. With no clear goal, either immediate or future, discovery exploration can be defined as most successful when it reveals all the features of the interface.

2.2.1 Defining Exploration Efficiency

Discovery-oriented exploration involves moving through the space and gathering information. We consider these two processes separately, beginning with algorithms for movement. The best algorithms for discover-oriented exploration would that provide maximum coverage of the space at minimum cost. How should "coverage" and "cost" be defined? We begin by defining some basic terms:

A	=	number of arcs in the exploration space
a	=	total number of arc traversals during an exploration
a'	=	number of arcs traversed once or more during an exploration
N	=	number of nodes in the exploration space
n	=	total number of node visitations during an exploration
n'	=	number of nodes visited once or more during an exploration

The values for N, n, and n' are defined to exclude the node where the exploration begins.

To build a complete map of the exploration space, the explorer needs to visit every node at least once and traverse every arc at least once. (The explorer can only discover an arc's target by traversing the arc.) We define coverage to be the number of arcs and nodes visited once or more.

coverage: $K = a' + n'$

The cost of exploration is defined to be the total number of actions, i.e., the total number of arc traversals.

cost: $c = a$

Using these definitions we can express a simple measure of the effectiveness of an explorer's efforts.

exploration efficiency: $E = K/c$

Under this definition, the efficiency rating increases as the cost of the exploration decreases or the coverage improves.

2.2.2 *Explorability of Different Spaces*

What can be said about the range of possible values for E? Clearly, they depend on the structure of the space and the path by which it is explored. The maximum E attainable for a given space will be referred to as the space's explorability.

explorability: E_0 = maximum attainable E for a given space

The definition of exploration efficiency can be used in defining an absolute upper bound on explorability. The highest exploration efficiency would result if each arc were traversed exactly once ($a = a' = A$, for minimum cost and complete arc coverage), and each node visited at least once ($n' = N$, for complete node coverage). Substituting into the definition of exploration efficiency yields:

upper bound on
explorability: $E_0 <= (A + N) / A$

$<= 1 + N/A$ Eq 2.1

The value of this expression depends on the ratio of nodes to arcs. A space with a high ratio will offer the potential for a higher exploration efficiency. But the requirements of exploration put an upper bound on this ratio. If a node is to be reached during exploration, it must be connected to the rest of the graph by at least one arc. This forces $A >= N$, so that $N/A <= 1$. Substituting into Equation 2.1:

upper bound on
exploration efficiency $E_0 <= 2$ Eq 2.2

Figure 2.2 shows a graph offering explorability of 2. The graph maps to a user interface for a trivial system. Each command in the system yields a new state, but only a single sequence of commands is available.

How low can the value for E_0 fall? In the space shown in Figure 2.3, exploration of each of the many arcs leading out of node C must be preceded by traversal of the arcs

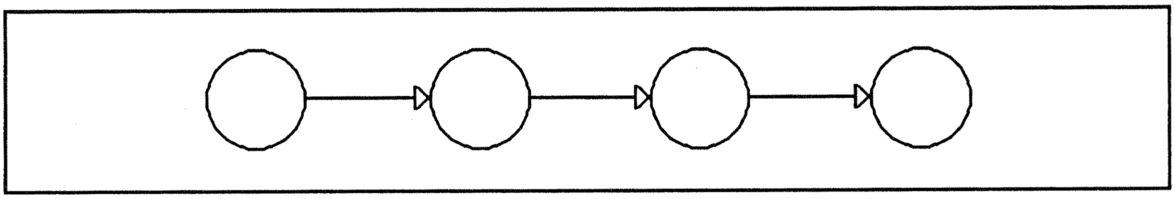


Figure 2.2. An exploration space that can be explored efficiently (E_0 = 2).

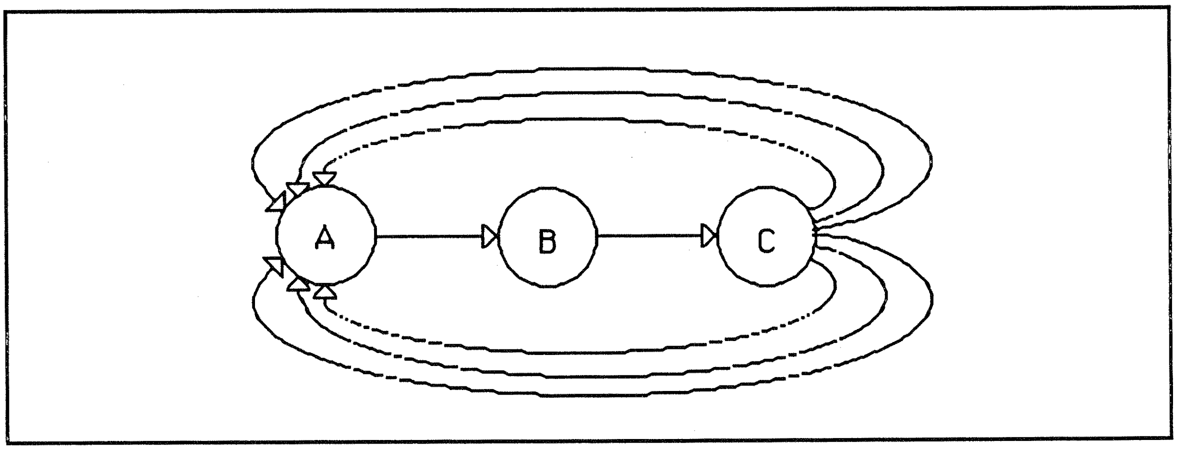


Figure 2.3. An exploration space that cannot be explored efficiently (E_0= approx 0.5).

from A to B and B to C. The E_0 value for this graph is less than one, about 0.5. By increasing the number of nodes and arcs intervening between A and C, we could force E_0 arbitrarily close to zero.* This establishes a lower bound for explorability.

range of possible explorabilities: $0 < E_0 <= 2$ Eq 2.3

The zero-to-two range of possible values for E_0 indicates that changing a system's underlying structure can have a significant effect on a user's efforts to explore the system. A space with $E_0 = 2$ can potentially be explored with 1/10 as many actions as a space with $E_0 = 0.2$. The next section considers whether this potential can realistically be achieved: Are there general algorithms that can yield optimal exploration efficiency of spaces having a high E_0 ?

2.2.3 Algorithms for Discovery-Oriented Exploration

For any exploration space there trivially exists one algorithm that achieves optimal exploration efficiency of the space. That algorithm, which may be difficult to discover, is simply a list of the arcs in the order that they must be traversed. However, the more interesting question is whether there are general algorithms that apply to many spaces. General algorithms for searching or traversing graphs and trees have been investigated at length in the contexts of data structures (e.g., Aho, Hopcroft, and Ullman, 1983) and artificial intelligence (review in Korf, 1988). These investigations suggest general forms for exploration algorithms.

Because previous investigations have considered search and tree-spanning procedures for computer implementation, the algorithms often make implicit assumptions that would give the explorer more power than is appropriate for our model. For example, the algorithms may assume that movement within a space is constrained only by information availability, not by the arcs themselves. That is, after traversing an arc from node A to B in Figure 2.3, an algorithm might return to node A to investigate other arcs leading out of it, even though there is no arc returning from B to A. Many of these algorithms also involve "marking" nodes as visited, something that may not be possible during interface exploration. We take several algorithms, describe their implicit and explicit requirements, and analyze their effectiveness for exploration.

* This describes the exploration space of a typical adventure game, where a series of choices must be made to reach a given depth, and all but one or two moves at that depth will throw the player back to the start of the game.

2.2.3.1 *Depth-first exploration*

Depth-first search is a well known technique that can be modified to produce an efficient exploration algorithm. The depth-first procedure can be easily understood for the special case in which the graph is a tree. The algorithm begins at the root and immediately moves outward to a leaf-bearing twig. Each leaf on the twig is visited exactly once, after which the traversal backs up a single level and moves on to the next twig, visits it in depth, and continues to move on. For the more general case of a graph, the algorithm simply avoids arcs leading to nodes that have already been visited; the resulting pattern of visitations describes a depth-first spanning tree, which includes all nodes of the original graph.

The standard depth-first algorithm assumes that the graph traverser can recognize an arc that leads to a previously visited node and avoid that arc. Since we want to try all the arcs (all the actions in the interface), and since predicting the target node of an arc is assumed to be impossible, we modify the algorithm so the explorer actually traverses each arc, returning immediately if the node entered has already been visited. An exploration using this algorithm is shown in Figure 2.4. The algorithm ensures that the explorer enters every state at least once and examines each arc exactly once. It is, therefore, optimally efficient as defined in the previous section.

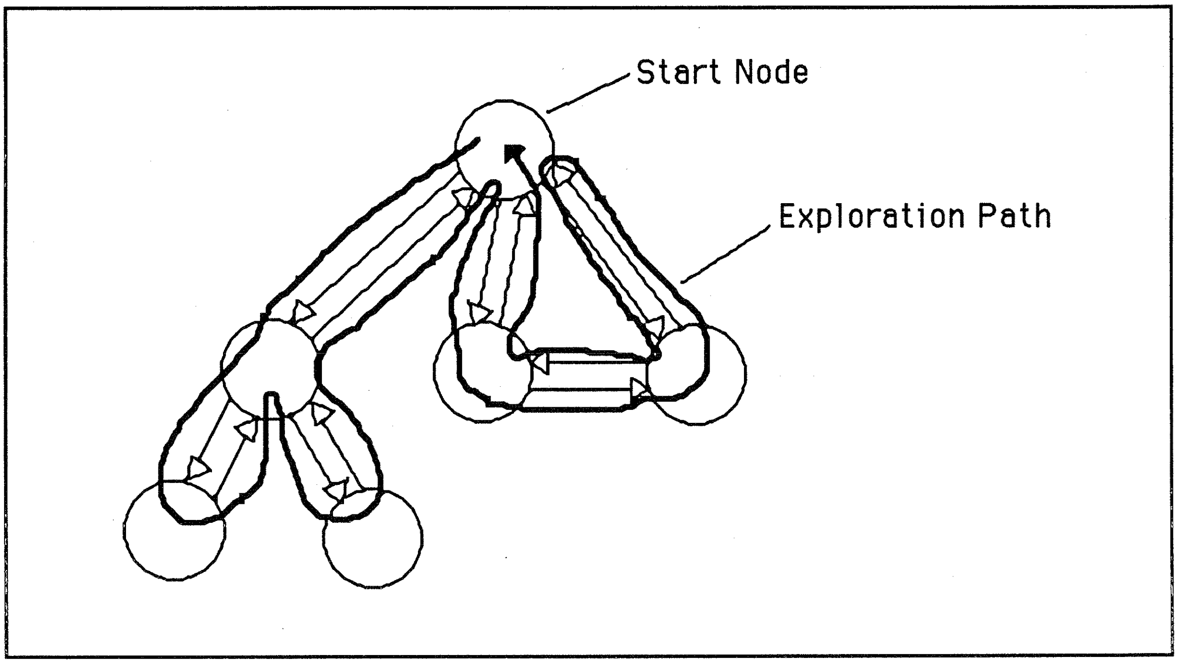


Figure 2.4. An exploration guided by the depth-first algorithm.

The algorithm has several implicit requirements: (1) the explorer must be able to recognize states as having been previously visited; (2) the explorer must be able to recognize arcs as previously traversed; (3) every arc from some node A to B must be paired with a return arc from B to A, and the explorer must always be able to recognize the return arc (for the user interface, the return arcs may be "undo's" or they may be other actions that cancel the effect of the previous action, such as setting the font back to plain text after setting it to italic). Because each node must be connected to the rest of the graph by a pair of arcs, $A = 2N$, and the exploration space required by the algorithm offers a maximum exploration efficiency of $1 + N/2N = 1.5$. The depth-first algorithm achieves this efficiency.

Although it is optimally efficient for exploration spaces that satisfy its requirements, depth-first exploration requires considerable activity for the interfaces commonly found on modern systems. Consider a very simple Macintosh interface with five items on the top-level menu bar, five commands on each pull-down menu, and a five-item dialog box called up by each of those commands. Complete depth-first exploration of this interface would require 250 actions, 125 of which are not undo's. (Actually, exploration of a standard Macintosh interface with the depth-first algorithm would not be strictly possible, since undo doesn't bring back the dialog box, and cancelling the dialog box doesn't bring back the menu that led to it.)

2.2.3.2 Breadth-first exploration

In breadth-first traversal of a graph, all nodes connected by a single arc to the start node are investigated on the first pass of the algorithm. On each subsequent pass, the algorithm probes one layer deeper, investigating all nodes that are one arc away from nodes visited during the previous pass. This procedure assumes that the traversal can always jump back to any node that has been previously visited and probe one level deeper from that point. In AI search applications, this requires that the state represented by every visited node be maintained in the search program's memory, which makes breadth-first search a memory-intensive procedure.

For exploration as it has been defined here, standard breadth-first search is simply impossible. The explorer cannot jump back to an arbitrary node that was previously visited. Only nodes connected by arcs to the node in which the explorer is currently found can be directly accessed, and even in these cases the destination of the connecting arc is not known until it has been explored.

2.2.3.3 *Depth-first exploration with iterative deepening*

Depth-first search proceeds all the way to the outermost leaf of a tree before turning back to investigate another leaf, twig, or branch. For many AI problems this is problematic. The depth of search on some or all of the branches may be too deep for computation in the time available, as in search of the game tree for chess. Another potential problem is that the node sought may lie lie at a very shallow depth on a branch that is not searched early on. Again using chess as an example, the algorithm could waste a great deal of time investigating the long end game that follows a pawn move, before going to investigate a queen move that would give an immediate win.

An alternative to depth-first search is depth-first search with iterative deepening (Korf, 1985). This algorithm combines some of the features of depth-first and breadth-first search. In DFID, the first pass of the search is identical to depth-first search, but only to the depth of one arc away from the start node. The second pass performs a depth-first search from the start node to depth two, the third pass to depth three, and so on. Unlike breadth-first search, each pass starts fresh, using none of the state information collected during the previous passes.

The DFID algorithm can also be used for exploration. It requires an exploration space with the same characteristics as for depth-first exploration, so the optimal explorability of the space is 1.5. But DFID is clearly not an optimal exploration algorithm, since each pass repeats all the arc traversals and node visitations from the previous pass, covering ground that has already been explored. The exact exploration efficiency of DFID will depend on the structure of the space. Assume a space in which every non-leaf node has b arcs branching out of it (and b return nodes leading back in). Let the space have a depth of d from the start node. Then DFID will traverse $2b$ arcs on the first pass, $2b + 2b^2$ on the second pass, $2b + 2b^2 + \ldots + 2b^d$ on the final pass. The sum of all traversals will be $2\,(db + (d\text{-}1)b^2 + (d\text{-}2)b^3 \ldots + b^d)$. The total coverage will be the number of arcs, which the same as the number of traversals on the final pass, plus the number of nodes, which is half the number of arcs. The exploration efficiency of DFID, then, will be

$$\text{efficiency of DFID:} \quad E = K/c$$

$$= \frac{3(b + b^2 + \ldots + b^d)}{2\,(db + (d\text{-}1)b^2 + (d\text{-}2)b^3 \ldots + b^d)} \qquad \text{Eq 2.4}$$

Table 2.1

Exploration Efficiency of DFID (from Equation 2.4)

	Branching Factor				
Depth	1	2	3	4	5
1	1.5	1.5	1.5	1.5	1.5
2	1.00	1.13	1.20	1.25	1.29
3	0.75	0.95	1.08	1.17	1.22
4	0.60	0.87	1.03	1.14	1.21

For $d = 1$, Equation 2.4 gives a value of 1.5, the same efficiency rating as depth-first exploration. Table 2.1 shows values for a reasonable range of depths and branching factors. The table demonstrates that, for branching factors that are reasonable in user interfaces (five or more menu items is not unusual), DFID never falls far below the optimal efficiency offered by depth-first exploration. This suggests that DFID might be a reasonable approach for task-oriented exploration of an interface, where an acceptable solution, like the winning pawn move, might be discovered at some shallow depth.

2.2.3.4 Random exploration

The random algorithm for exploration selects an arc at random, traverses it, and repeats the process until stopped by external interruption or some defined criterion, such as number of traversals. The random approach is interesting because it approaches a lower bound for control sophistication. It is appropriate to say "approaches," not "describes," since truly random selection of an alternative may itself be difficult. The algorithm is also interesting because its effectiveness decreases as exploration proceeds. In the space shown in Figure 2.4, random selection is guaranteed to choose an unexplored arc for the first and second traversals, and it has at least a 50 percent chance of choosing unexplored arcs for the third and fourth traversals. Eventually, however, the space will be mostly explored, and random selection will have a poor chance of selecting the few unexplored arcs. Repeated traversals of previously traversed arcs might be an appropriate stopping criterion for the algorithm.

The practical effectiveness of random exploration will vary with the degree of coverage desired. For example, the random algorithm might have an average efficiency of 1.0 in exploring 75 percent of a certain exploration space. For spaces requiring complete coverage, random exploration would be a poor choice, but for partial coverage, it might be a reasonable approach to other constraints, such as information management (discussed below).

The same approach suggests an analysis of explorers who implement an exploration algorithm with random error. Consider a depth-first explorer who has a 10 percent chance of missing any arc. In a space with a depth of three, for example, we can expect the explorer to miss 10 percent of the arcs (leading to 10 percent of the space) at the top level, plus an additional 10 percent at each of the subsequent levels, for a total miss rate of $0.1 + 0.1*0.9 + 0.1*0.81 = 0.271$. This is approximately the base error rate times the depth. But with DFID, the imperfect explorer of the same space investigates the top level three times and the second level twice, yielding a total error that is not much greater than the base rate.

The structure of the space can also affect the total error rate. Multiple arcs leading to a single node reduce the chances that the node will be missed, while hiding a major portion of the space behind a single arc increases the chance of missing that entire subspace.

2.3 Task- and Problem-Oriented Exploration

Recall from Chapter 1 that "task-oriented" exploration is similar to discovery-oriented exploration, but with the user's knowledge contributing more to the high-level task goals. The example given for task-oriented exploration was the design of a newsletter using a new page layout system. "Problem-oriented" exploration is the situation in which a detailed task goal has been set but the interface actions needed to achieve it are unknown. An example of this would be specifying bold font for a line of type in a new word processor. As discussed in Chapter 1, these categories are not sharply delineated.

2.3.1 Using Opportunistic Goals to Guide Exploration

Problem and task-oriented exploration are important topics in their own right. However, they gain additional importance as potential solutions to the difficulties of exhaustive exploration uncovered in the analysis up to this point. Those difficulties can be summarized by observing that an discovery-oriented exploration of an entire interface is likely to exhaust the user long before it exhausts the possible combinations of states and actions afforded by the system. A clear way to reduce those difficulties is to explore only those combinations of states and actions that seem somehow "productive," or to put it another way, to limit the exploration to those actions that might satisfy some reasonable goals.

Shrager's protocols of students learning the Big Track toy suggest that learners may structure their explorations in exactly that way. In the absence of clear pre-existing goals, Shrager's subjects combined background knowledge with options offered by the interface to suggest microtasks that the interface and system might support. They then sought for solution paths that achieved those microtasks.

In general, then, a user faced with the need to explore a new system may find it effective to create "weak," opportunistic (Hayes-Roth and Hayes-Roth, 1979) goals. These weak goals can then drive a limited exploration. These goals should appear suddenly, triggered by the interface, and they should be dropped if continued exploration fails to satisfy them, or if more attractive goals appear as the interface and system are revealed. Beyond simply limiting the search to a manageable size, the goal-oriented approach should direct the users' efforts to parts of the interface which, because they match his or her background knowledge and understanding, are most likely to be useful in the future.

2.3.2 Goal-Oriented Exploration Compared to Search

Goal-oriented exploration is very similar to search. As with search, the exploration can be terminated when the stated goal matches the state of the world. However, it can also be terminated when extended efforts begin to suggest that the goal cannot be achieved. Certain features of the interface can make the exploration more or less difficult, depending on the algorithm. One critical feature of the interface, if the algorithm recognizes it, is the semantic information content of the control labels.

In the optimal case, the control labels would guide the user unerringly through the task. This can only occur if (1) the user's overview of the task structure is similar to the structure actually offered by the interface (i.e., the user can't be trying to load the spelling dictionary when the system wants the user to select the section of the document to be spell-checked) and if (2) the user's representation of the steps of the task, expressed cognitively as a sequence or collection of subgoals, is sufficiently similar to the control labels that the user can recognize a uniquely correct control at each step.

It seems likely that most goal-oriented exploratory situations will be less than optimal. The user will have some misunderstandings as to the task structure, and some of the control labels will fail to unambigously match the user's immediate goals. In these situations, the user must still explore the interface, but the exploration need not be exhaustive. It can be limited, at least initially, to those controls whose labels appear to match the current goals.

2.3.3 An Algorithm for Goal-Oriented Exploration

The best approach for goal-oriented exploration depends strongly on the semantic value of the control labels. As noted above, the best possible situation is one in which the labels completely guide the user's selections, so searching to the end of a single path achieves the current goal. The worst possible situation is one in which the labels (and the changes in state that occur as paths are followed) contain no useful semantic information. In this case, exhaustive exploration must occur until the goal is achieved, For exhaustive exploration, as discussed earlier, the DFID algorithm is an effective algorithm that approaches optimal efficiency for many interfaces. It has the additional benefit of achieving shallow coverage of the interface quickly, which may be appropriate in an interface where the solution (or clues to the path toward a solution) may be discovered at or near the top level.

The DFID algorithm can be modified for slightly greater usefulness by choosing to first deepen those paths that the interface allows to be deepened quickly (as opposed, for example, to a path that requires the system to pause for several seconds to perform a search). This modification would have no impact in fully exhaustive search, but it is sensible where a goal has been set, since search can terminate as soon as the goal is achieved.

For task-oriented situations, where less-than-exhaustive search is expected to suffice, the DFID algorithm can be further modified to take into account the semantic content of the labels and the semantic content of the feedback to actions as they are attempted. The modified algorithm, "guided DFID" (gDFID) is similar to the beam-search version of breadth-first search (Winston, 1984) and to the IDA* algorithm that describes a form of depth-first search (Korf, 1988). In gDFID, paths that appear to be relevant to the current goal are expanded first. If the search continues to fail, it is extended to items that appear irrelevant to the stated goal. This extended search first investigates those controls whose effects cannot be predicted by their labels and feedback; then, if this fails, moves on to investigate controls with labels and feedback whose semantic content actually contradicts the current goal.

2.4 Information Management During Exploration

As exploration proceeds, the explorer is faced with two related information management tasks. The information revealed by the exploration must be recognized and possibly retained, and the information needed to control the exploration algorithm must be maintained.

It is useful to describe the kinds of information that exploration may reveal and consider what an explorer might remember. (1) In task-oriented exploration, the explorer may consider features of the system only long enough to determine whether they are applicable to the current task, but retain nothing of value after the exploration is done. (2) In either discovery- or task-oriented exploration, the explorer may learn that certain functions are available in the software, but not remember the command sequence to invoke those commands. In the model, this maps to learning the contents of the nodes but nothing about the arcs. (3) Finally, in any exploration the explorer may learn that functions are available and also learn how to access those functions. That is, knowledge of the arcs (source, label, and target) and the nodes (content) may both be retained.

The information needed to control the exploration depends on the algorithm. The random algorithm requires no retention of control information from one action to the next. Depth-first exploration, on the other hand, requires that extensive control information be maintained. The explorer must remember which nodes have been visited, which arcs have been traversed, and which arc should be used to back out of every node that has not been completely explored. The DFID and gDFID algorithms require the same information as for depth-first exploration, along with a record of the depth to which exploration has currently progressed.

The two information-management tasks interact. An explorer who learns the label and target of each arc as it is traversed (i.e., the name and effect of each command) can use this information to control depth-first exploration. Arcs whose target is known have been visited need not be accessed again.

It is evident that the requirements of information management in sophisticated algorithms exceed the capacity human memory, at least for exploratory activities that are to be performed at a reasonable rate. Rapid storage of information into short-term memory will handle fewer than 10 items of information: names of nodes visited, return paths, etc. Augmenting this storage with long-term memory is possible, but the time to store items in LTM is typically on the order of 10 to 30 seconds (Card, Moran, & Newell, 1983), which would unacceptably slow exploratory activities with most modern interactive interfaces.

In information-management terms, the gDFID algorithm for task-oriented exploration is a relatively low-cost approach, since it requires only that the explorer remember which of the "attractive" options have been examined at each level. Unattractive options need not be remembered, since exploration down those paths will be blocked on the next pass by the attractiveness criteria. Indeed, reevaluating the attractiveness of all options at each iteration may improve search efficiency, since options that seem inappropriate at first glance may become more attractive as additional information about the interface is accumulated.

2.5 Limits of the Model and of Exploration

The model presents a radical simplification of the exploration space defined by a complex computer system. Simplification is the essence of modelling, but it is important to recognize where the model fails to reflect critical factors involved in real exploration. Not only does this suggest weaknesses of the formal model, it may also suggest problems with users' real algorithms, which may reflect similar difficulties in building a mental model of the real world.

2.5.1 Mapping Difficulties

Attempting to map an actual user interface onto the model reveals several difficulties. First, and probably most difficult, is the question of how to define a "state" and a "command." In a menu-based system, menu items are clearly commands. But is each keystroke also a command? Does a new state result each time another number is entered into a spreadsheet or another character into a word processor? If it did, combinatorial expansion of states and commands would make exploratory learning a hopeless task. But there are some instances, such as adding the string "asdfghj" to a text file before running the spelling checker, where changes to data do effectively produce a new state.

This difficulty recalls our early efforts with the cognitive walkthrough procedure, in which we attempted to analyze interfaces at a keystroke level (Lewis, Polson, Wharton, and Rieman, 1990). The value of such a detailed analysis is limited to walk-up-and-use interfaces. When considering users with background experience in an established computer environment, we now analyze well-practiced action sequences, such as "select PRINT from the FILE menu" (Polson, Lewis, Rieman, and Wharton, 1992).

A similar approach allows a meaningful mapping between the formal model of exploration and a real system, given that the user is not a complete novice. But the difficulty in mapping points to potential problems for the user. In order for the user to perceive a discrete space of commands and states, the interface must adhere closely to established conventions. Commands that involve actions that usually have no special effect, such as clicking at a blank area of the screen, can't be discovered unless the user engages in virtually unbounded exploration. Similarly, state changes that affect the behavior of commands will be difficult for the user to recognize without cues. Such cues might include menu items that are disabled when inappropriate or messages that appear when commands are attempted from an inappropriate state.

2.5.2 Differential Information Values

As defined, the expression for coverage assigns an equal value to every node and every arc. This seems overly simplistic when compared to the values an interface explorer might assign to the results of the exploration. Should traversal of paths be valued the same as visitations of states? Should all paths have the same value? Should multiple paths to a single state (multiple commands with the same effect) be given the same weight as discovery of a new state?

Consider the DFID exploration algorithm, in which as many as half of the arc traversals may represent undo commands. All the arcs are counted in the value for coverage, but after concluding that the undo command returns to the previous state, the user gains essentially no new knowledge from the repeated undos required by the algorithm.

Or imagine a user exploring a graphics program to produce a black-and-white graphic. The user can safely avoid the top-level menu for color and all the items under it. It is clear that some items of information can be more valuable than others simply because of the underlying structure of the exploration space.

2.5.3 Resources Not Recognized

The model suggests that exploration may be unachievably difficult. But this view may result from failing to consider all the informational resources at hand. Remembering all the "arcs" that have been traversed (or even all the attractive arcs, in the case of gDFID) does indeed seem to exceed the capacity of human memory, but in many cases the user may only need to remember a pointer: all menubar items to the left of "Document" were investigated, or simply all menubar items to the left of the current cursor position. Cues such as these are inherent in display-based interfaces, potentially lending them an explorability that command-line interfaces lack. Display-based interfaces also support the use of recognition memory, and an algorithm utilizing this approach has been described by Howes (1994). Finally, expert users may make use of "long-term working memory" (Kintsch & Ericsson, 1991), to rapidly encode and recall much more information about the interface than a novice user can handle.

2.6 Practical Difficulties Exposed by the Model

Several conclusions can be drawn from the model of exploration presented in this chapter.

- Recognizing states and commands is a fundamental problem.
- Combinatorial expansion makes exhaustive exploration almost overwhelmingly difficult, even with clearly delimited states and commands.
- The both the structure and the perceptual form of an interface can have a significant effect on the attainable exploration efficiency.
- Information-management needs may make sophisticated exploration algorithms unusable by human explorers.

The next chapters will investigate whether users can overcome these difficulties, and if so, how.

Chapter 3

DIARY STUDIES OF REPRESENTATIVE USERS

This chapter analyzes the one-week logs kept by the fourteen field informants. The logs show a wide variety of tasks and work habits, but they present a common picture of how task-oriented problems are solved when they arise: by trying things out in the interface, looking at paper manuals, and asking other people for help.

The previous chapter described some of the difficulties theoretically associated with the exploration of computer systems. Have users of current interactive systems found ways to overcome these difficulties? Or have they decided that exploration is hopelessly difficult? If users do engage in exploration, under what circumstances is this behavior most likely to occur?

To answer these questions, and to build a foundation for laboratory investigations and further theoretical work, a field study was performed of users' exploratory and learning behavior in the course of their ordinary workday. The technique used was the diary study (Ericsson, Tesch-Römer, and Krampe, 1990; Rieman, 1993). The "grain size" of the data exposed by the diary study is relatively large. Events that take several minutes are reported, but the low-level detail of those events is not clearly exposed. Thus, the field study will give evidence of the existence and context of exploration, but not its structure. User behavior at a finer grain size will be the subject of Chapters 5 and 6.

3.1 Method

3.1.1 Informants

Fourteen informants participated in the study, seven male and seven female. The participants are identified as Informants 2 through 16. Informant 1 was a pilot subject whose data is not reported, and there was no Informant 13. Individual participants are always referred to with male pronouns, to help preserve anonymity. The Informants all volunteered their time. The informants were selected to provide a broad range of experience and work demands, as shown in Table 3.1.

Table 3.1
Informants' Background and Experience Ratings

ID*	Expt†	Op. Sys.**	Computer Use	Position; Duties	Background
2	1	Unix/ Work-station	clerical	secretary; general	On-the-job and classroom training on Unix-based and dedicated word processing, spreadsheets.
3	2	Mac/ VMS	clerical	secretary; general	On-the-job training, some classes, on dedicated word processors, PC and Macintosh-based word-processors and spreadsheets.
4	5	Work-station /Unix	work/ research	Ph.D. student in computer science; AI research, writing	Industry experience in program development and support. Extensive use of minicomputers, personal computers, AI workstations.
5	5	Mac/ Unix	work/ research	Ph.D. student in computer science; AI research, writing	Industry experience in program develpment and support. Experience with Unix, PC's; learning Macintosh. Has taught computer science courses.
6	1	Mac/ PC	clerical	undergraduate in engineering, clerical assistant duties; home-work	Home experience with PC's. Work experience with word processing and spreadsheets on Macintosh. Some programming training.
7	5	Unix/ Work-station	work/ research	faculty member in computer science; AI research, teaching	Extensive academic experience with Unix systems and AI workstations. Limited PC/Macintosh background. No industry experience
8	4	Mac/ Unix	primary tool	research faculty in cognitive science; cognitive modelling, empirical research	Master's degree in computer science; has taught computer science courses. Macintosh user and Lisp programmer. Research focus on programmers, not software applications.

Table 3.1 (Cont.)

9	3	Mac/ PC	clerical	undergraduate in pharmacy, clerical assist.; homework, assigned duties	Work and home experience with several programs on PC and Macintosh. High-school programming courses. Aggressive learner of new programs
10	5	Unix/ Mac	work/ research	Ph.D. computer science researcher in industry; program development	Industry experience in program development and support. Extensive use of minicomputers, personal computers, AI workstations. Has taught computer science courses.
11	4	PC/ VMS	science support	faculty member in social sciences; research, teaching	Extensive experience with personal computers, both command line and graphical interfaces. Relies heavily on databases, word processors, and presentation software for teaching and research. Programs in database, system command language.
12	3	Mac/ VMS	science support	Ph.D. student in psychology; empirical & bibliographic research	On-the-job experience with minicomputer statistics packages, Macintosh spreadsheets, word processing, and graphics. Some programming training, occasional programming for research in statistics programs, HyperCard.
14	4	PC/ Mac	primary tool	financial analyst in industry; data analysis, predictions	Master's level training in information systems. Programming courses in FORTRAN, Pascal. Extensive experience with PC's and Macintosh, both as a user and support person. Frequent programmer in database and spreadsheet packages.
15	2	Mac/ VMS	science support	faculty member in psychology; empirical research, teaching	On-the-job experience with Macintosh word processing, graphics, and statistics. Some experience with similar programs on window-based PC's. E-mail on minicomputer. Has avoided learning to program.
16	4	PC/ Mac	primary tool	financial analyst in publishing; data analysis, workflow investigations, training for job sharing	Courses in programming (FORTRAN, COBOL, Pascal) and finance. Extensive experience evaluating Macintosh application software in a development environment. Heavy user of spreadsheets and databases. Involved in user support.

* ID 1 was a pilot subject whose data is not reported; there was no ID 13.
† Primary/secondary operating system; PC includes both MS-DOS and MS Windows.
** See Table 3.2 for explanation of experience ratings.

3.1.1.1 Computer use distribution of informants

Four informants, identified by the keyword "clerical," in the computer-use column of the table, were selected as representative of the large population who uses word processors and spreadsheets in an office environment. Two of these users were professional secretaries in university academic departments, one using a Unix-based system, the other working with Macintosh equipment. Two were undergraduate student clerical assistants, doing secretarial and general office work as well as their classwork. None of these users had extensive computer science training or skill, although the secretaries had professional level skills with the software they used in their work.

Three informants were selected to yield insights into the work of scientists who rely on computers for text processing, graphics, and some data analysis, activities that are in support of their main research interests. Two of these informants were faculty members in departments outside computer science (a social science and psychology), neither of whom was involved in computer modelling of cognition or other research that demanded extensive computer work. A third informant was a doctoral student in psychology, again in an area that did not demand a high level of computer skill.

Another three informants, whose computer use is described as "primary tool," represented the growing force of workers for whom the work environment is not simply improved but is entirely defined by the presence of computers. Two of the informants worked in business, where they did financial modelling and analyis that would have been impractical or even impossible before the advent of PC-based spreadsheets and databases, often in communication with mainframe accounting packages. These informants were both highly skilled users and programmers within the applications they used, but neither was a trained software developer. The third informant in this group was a cognitive science researcher, with a strong background in computer science, but with research activities in which computers were a tool for empirical analysis and cognitive modelling, not a topic of investigation in themselves.

Finally, four of the informants were highly experienced computer scientists, including a faculty member, two graduate students nearing completion of their Ph.D.s, and an industry researcher with a Ph.D. and several years software development experience. These users perform work and research in the field of computers themselves. It was expected that the most extensive exploratory activities would be seen in this group.

3.1.1.2 Experience distribution of informants

The informants' experience with computers ranged from novice users who had worked with only one or two word processors to professional Ph.D. computer scientists with experience on systems as simple as a Macintosh and as complex as a Symbolics workstation. The years of experience for each user were difficult to determine, and the experience ratings defined more in terms of breadth of experience than depth. Table 3.1 lists the ratings for each user, and Table 3.2 provides a key to the meaning of the numerical ratings. The novice (narrow) category described the background of two of the users, who had only worked extensively with a few word processors, with some additional exposure to other programs such as spreadsheets and graphics. Two more users fell into the novice (broad) category, having more training and more extended experience with software in addition to word processing. The intermediate category, which also included two informants, was defined to be the lowest level that used

Table 3.2

Key to Experience Ratings

Rating	Number of Inf's	Description	Defining Experience
1	2	Novice (narrow)	Work experience limited to one or two word processors, e-mail, possibly a spreadsheet; no use of programming.
2	2	Novice (general)	Work experience with several types of software application, some formal training, no use of programming in job.
3	2	Intermediate	Work experience with many software applications and more than one system, programming training or experience, programs infrequently.
4	4	Advanced	Work experience with many software applications and systems, frequent use of programming in work (including traditional languages or application languages such as spreadsheet macros, complex database queries, system command files), has supported other users professionally.
5	4	Expert	Professional programmer, familiar with many systems, extensive formal training in computer science, has taught or supported other users.

programming (including operating system command files, data base queries, and other application-specific languages). Four informants fell into the advanced category, which required experience with a variety of systems and frequent work-related use of programming. Finally, the expert category included four professional computer scientists, all with Ph.D.-level training (two Ph.D.'s in progress) and all with industry experience in program development.

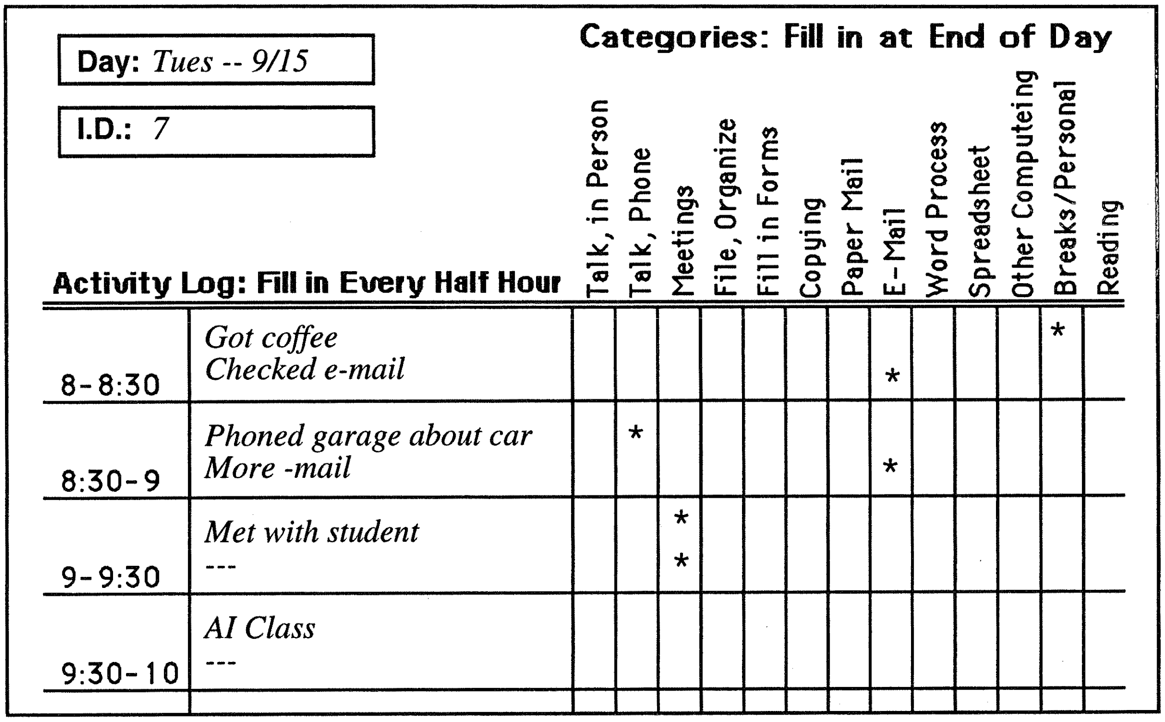
3.1.1.3 Operating system distribution of informants

In addition to having a wide range experience, the informants also used a variety of computer operating systems as their primary system. Systems represented included Macintosh (7 users), PC's (with or without MS Windows; 3 users), Unix or VMS (3 users), and high-end scientific workstations (i.e., Sun, Symbolics; 1 user). Systems on which users had secondary access or significant past experience, typically for e-mail purposes or on a home system, included Macintosh (3 users), PC's (2 users), Unix or VMS (7 users), and high-end scientific workstations (2 users). For some users it was difficult to say which was the "primary" system; in these cases the system listed is the one in which the user reported the most computing during the log week.

3.1.2 Materials

Informants maintained their daily log of activities on a log sheet of the form shown in Figure 3.1. The sheet was on 11-inch by 14-inch paper, and the hours on the left side corresponded to the informant's typical working hours, with a few hours extra. Usually this was from 7 a.m. to 7 p.m., but some subjects worked a later schedule and used a sheet that ran from 10 a.m. to 10 p.m. In addition to the log sheet, informants were supplied with a stack of blank "Eureka Slips," as shown in Figure 3.2. These were printed on various bright colors of paper and held together with a colorful spring clip. The colors were intended to make the slips more visible and increase the likelihood that informants would remember to fill them in when appropriate.

The materials were designed to focus on learning, without specifically narrowing the issue to exploratory learning. There was no specific category on the log sheet in which to record time spent exploring computer systems. The sheet included an "other" category as well as categories for common systems (word processing, spreadsheets, etc.), and these were intended to cover all computer activity. The Eureka slips narrowed the focus somewhat, being clearly concerned with learning, but they were broad enough to cover many kinds of learning activity, including both task-oriented and task-free exploration.



Day: *Tues -- 9/15*

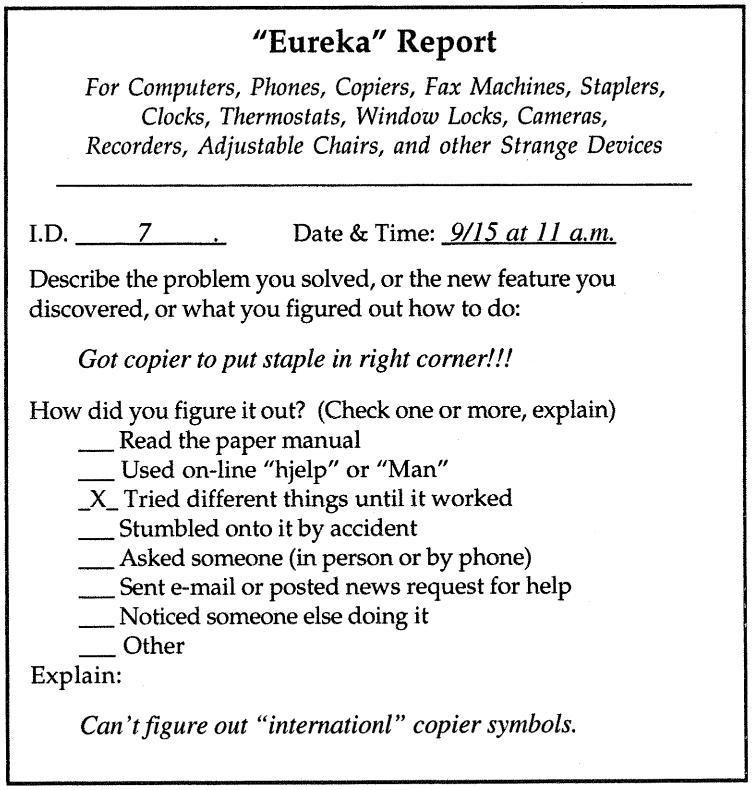
I.D.: *7*

Categories: Fill in at End of Day

Activity Log: Fill in Every Half Hour

		Talk, in Person	Talk, Phone	Meetings	File, Organize	Fill in Forms	Copying	Paper Mail	E-Mail	Word Process	Spreadsheet	Other Computeing	Breaks/Personal	Reading
8-8:30	*Got coffee* *Checked e-mail*								*				*	
8:30-9	*Phoned garage about car* *More -mail*		*						*					
9-9:30	*Met with student* ---			* *										
9:30-10	*AI Class* ---													

Figure 3.1. The beginning of a diary log sheet for one day. The participant records activities, and the researcher assigns categories during the end-of-day debriefing.



"Eureka" Report

For Computers, Phones, Copiers, Fax Machines, Staplers, Clocks, Thermostats, Window Locks, Cameras, Recorders, Adjustable Chairs, and other Strange Devices

I.D. ___7___. Date & Time: *9/15 at 11 a.m.*

Describe the problem you solved, or the new feature you discovered, or what you figured out how to do:

Got copier to put staple in right corner!!!

How did you figure it out? (Check one or more, explain)

___ Read the paper manual
___ Used on-line "hjelp" or "Man"
_X_ Tried different things until it worked
___ Stumbled onto it by accident
___ Asked someone (in person or by phone)
___ Sent e-mail or posted news request for help
___ Noticed someone else doing it
___ Other

Explain:

Can't figure out "internationl" copier symbols.

Figure 3.2. A "Eureka" slip, for noting successful or attempted learning events.

3.1.3 Procedure

The investigator gave a brief description of the log materials to the informants at the time they were recruited. A log period was scheduled at that time. As much as possible, the log period was scheduled to be a week of "ordinary" activity for the user. A week when an informant would be out of town for a conference, for example, would not be logged (unless travelling to conferences was a common activity for that user); but there was no effort made to prefer or avoid times when the user had just started work with a new piece of software.

Immediately before the log period began, the investigator met with the informant and explained how the logs were to be used. Informants were to fill in the left-hand side of the log sheet as each day progressed, using their own terminology for the work they were doing. To protect their privacy, they were allowed to mark any period as "breaks/personal" and provide no further information. Informants were instructed to fill in a Eureka slip whenever they learned something new about their computer system or some other equipment in their work. They filled in the slips completely, including the description of the problem, the strategy used to solve it, and any additional comments. Informants were also asked to use Eureka slips to record failed attempts to solve problems.

At the end of each log day, the investigator met with the informant (in person for all but two informants, and over the phone for those two (10, 14)). The investigator spent 10 to 20 minutes talking over the day's activities and assigning them to the categories on the right-hand side of the log sheet. The day's Eureka slips, if any, were also discussed briefly during this meeting, and the investigator often made additional notes on the back of the slips, or occasionally corrected a strategy assignment. If logged activities revealed a learning episode that had not been recorded on a Eureka slip, the investigator and the informant would fill one in for that episode.

3.2 Results: Logged Time

This chapter presents summaries and analysis of the log data. The data presented in this chapter include only log time not marked as breaks/personal.

All but two of fourteen informants kept their logs over a consecutive 5-day period. One informant (3) kept a log over 4 1/2 work days plus 1/2 work day; for another (10), one day of the five was eliminated from the data because the log was only partially and inaccurately completed. For most informants, the log skipped weekends and holidays, although some informants worked over the weekend and logged that period.

3.2.1 Overview

An overview of the logged time is given in Table 3.3. The mean number of hours logged, after personal time was subtracted, was 34, ranging from 27 to 39.5 hours per informant. Within those 34 hours, informants spent an average of 17.3 hours, or 50.9 percent of their time, working with computers. For the purposes of this analysis, computers were defined narrowly to include PC's, minicomputers, and mainframes. They did not include phone systems, VCR's, fax machines, copiers, and similar hardware that typically has a dedicated internal microprocessor and a multi-function interface. The format of the log sheet made it difficult to accurately determine time spent in these activities, since individual sessions with these machines typically took much less than the log's 30-minute basic interval. Computing time for individual informants ranged from 1 hour (informant 6) to 34.4 hours (informant 5). As a percentage of their total log time, computing time for individual informants ranged from 3.5 (informant 6, 1 hour out of 28.5) to 91.7 percent (informant 8, 27.75 hours out of 30.25).

Table 3.3

Hours Logged During the Diary Study

	Total	Mean	StdDev	Min	Max
Hours Logged, Less Personal Time	476.5	34.0	4.27	27.0	39.5
Hours of Computing	242.6	17.3	8.41	1.0	34.4
% of Log Hrs Spent Computing	50.9	50.9	24.3	3.5	91.7

3.2.2 Time in Application Categories

As part of the log, informants recorded the category of computer application they were working with. That data is summarized in Table 3.4. The categories should be self-evident except for "special applications," which included mostly minicomputer-based software that was custom programmed for some business function. Operating system activities included disk formatting, directory management, and backups; working with batch or command files was categorized as programming. The news and on-line information category reflected mostly internet news reading, but one subject also spent time looking at the University of Colorado's on-line general information.

Table 3.4

Hours Logged in Software Categories During the Diary Study

Software Category	Total Logged			Percent of Informants' Computing Times			
	Hours	% Ttl	Inf's*	Mean	StdDev	Min	Max
Word Processing	87.5	36.1	12	37.7	30.6	0	88.5
Programming	28.5	11.7	4	7.6	25.9	0	97.3
E-Mail	26.0	10.7	9	15.3	22.1	0	60.2
Database	25.8	10.6	4	7.6	14.2	0	40.6
Special Applications	24.0	9.9	4	13.6	26.3	0	85.9
Spreadsheets	17.3	7.1	5	5.9	14.7	0	55.2
Graphics	11.5	4.7	4	4.0	9.4	0	33.9
Telecom (up/dnload)	9.0	3.7	4	2.4	5.0	0	16.7
Operating System	6.8	2.8	7	3.0	5.7	0	21.0
Games	3.5	1.4	1	1.4	5.4	0	20.3
News/On-Line Info.	2.5	1.0	3	1.2	2.5	0	8.7
Unknown	0.3	0.1	1	0.1	0.4	0	1.7
total	242.6		14				

*Number of informants who logged time in the category.

The left side of the table is a simple total of all times recorded, with a count of informants who reported spending time with each category. This is an indication of the study's application sample characteristics: if there was more time spent in one application category than another, then, all other things being equal, we could expect to see proportionately more learning events in that category. The category accounting for the most hours is word processing, which makes up 36.1 percent of the total computing hours logged. No other single category is a standout, although there is a range of 10 to 1 between the group of four categories that each make up about 10 percent of the remaining time (programming, e-mail, databases, and special applications) to the least popular application (network news and on-line information at 1 percent).

The right side of the table was developed by determining the percentage of each informant's total computing time that was spent in each category, then calculating the averages of those percentages. This is a rough indication of the sample population's work habits, and its most notable characteristic is its variance. The mean percentage for word processing, where informants on the average spent the largest proportion of their computing time, is 37.7 percent, but with a very great range. Two of the fourteen informants did no word processing at all. Of the remaining categories in Table 3.3, only electronic mail shows a mean percentage (15.3) and a number of informants (9) that even approaches a consensus.

3.2.3 Time Spent Exploring Computer Systems

As described above, the log sheets did not include a category for exploratory learning. Therefore, the record of this behavior is based on the investigator's daily debriefings and on the Eureka slips, described in greater detail in the next section. These data sources show no evidence of time spent by any subject in task-free exploratory learning of an interface. Informant 12 spent 15 minutes browsing the university's on-line information system with no set goal, but this was an exploration of the information, not of the interface. Informant 11, who had recently acquired new software, spent more than 5 hours doing task-oriented exploration of the software's capabilities, out of 15.5 hours total computing time. The tasks were small and artificial, and the informant admitted during the daily debriefings that his work-related need for the software really wasn't great enough to justify the time he was spending with it. Three other informants (5,10,16) were working with new software during the log week, but all of them structured their activities entirely around real, current tasks.

3.3 Results: the Eureka Slips

A total of 78 learning events were recorded on Eureka slips. Of these, 18 involved copiers, phone systems, or other equipment whose operation was not included in the calculation of each informant's "computing time." Because no background information was collected concerning these systems and users' related experience, those 18 Eurekas are not included in the detailed analysis that follows. Eurekas of all types are described in the Appendix.

The distribution of the 60 computer-related Eurekas across informants is summmarized in Table 3.5. One informant (5) reported 15 Eurekas; the counts for the other informants ranged from 0 (Informants 6 and 15, who both reported non-computer Eurekas) to 8 (Informant 11). As the table shows, the time spent in computing ranged from 1 hour to 34.4 hours. This information can be used to normalize the Eureka count for each informant, providing an indicator of learning events per computing time. The measure "Eurekas per 8 hours of computing" (E/8hr) was chosen as an easily comprehensible unit. For this measure, the range is 0 to 4.13, and the variance is proportionately somewhat less than for the raw Eureka count.

The E/8hr scores for three users stand out: Informants 5 (with a raw Eureka count of 15), 11, and 16 have scores of 3.48, 4.13, and 3.43, respectively. The E/8hr scores for all other informants are less than or equal to 2.30, with a mean of 1.19 (σ=0.8). This suggests that there may be two distinct types of users, or perhaps two distinct situations in which users find themselves. In either case, by far the most

Table 3.5

Hours and Eurekas Logged During the Diary Study

	Total	Mean	StdDev	Min	Max
Hours of Computing	242.6	17.3	8.41	1.0	34.4
Number of Eurekas	60	4.29	3.95	0	15
Eurekas per 8-Hr Computing (E/8hr)	1.73	1.73	1.28	0.00	4.13

Table 3.6

Eurekas per 8-hour of Computing, by Informant Categories

	Mean	Mean*	Values
Overall	1.73	1.19	
by Gender			
female	1.98	1.26	
male	1.48	1.14	
by Experience			
1 (novice)	.905	.905	(1.81, 0)
2	1.15	1.15	(2.3, 0)
3	2.31	1.41	(.62, 4.13, 2.19)
4	2.08	1.42	(1.73, 1.1, 3.43)
5 (expert)	1.72	1.14	(1.59, 3.48, .89, .93)
by Computer Use			
clerical	1.18	1.18	(1.81, 2.30, 0, .62)
science support	2.10	1.11	(4.13, 2.19, 0)
primary tool	2.08	1.42	(1.73, 1.1, 3.43)
work/research	1.72	1.14	(1.59, 3.48, .89, .93)
by Primary Operating System†			
Macintosh	1.47	1.14	(2.3, 3.48, 1.73, .62, 2.19, 0, 0)
MS-DOS/MS Windows	2.89	1.10	(4.13, 1.10, 3.43)
Unix/VMS	1.21	1.21	(1.81, .89, .93)
Workstation	1.59	1.59	(1.59)

* mean of all values except the three scores exceeding 3.0 E/8hr.
† not necessarily the system in which all Eurekas occurred.

prevalent case is that of the user who learns new things about a system infrequently, perhaps only two or three times a week for a user who spends as much as 50 percent of his or her time working with a computer.

In Table 3.6, the E/8hr scores of informants are broken down by gender, experience, type of work, and principal operating system. There are no notably large differences between the scores within any category, especially after the values for the three unusually high E/8hr scores are removed from the means.

The weak trend toward a greater number of learning events per 8-hours of computing by more experienced subjects is made more obvious if the Eurekas are categorized as simple or complex. An example of a simple Eurekas is learning how to select several files at one time on a Macintosh. And example of a complex Eureka is getting a simple "hello-world" program to run in a new data-base language. Clearly the complex Eureka involved learning many more individual facts than the simple one, although both were reported on single Eureka slips.

Table 3.7

Distribution of Eurekas, by Strategy and Informant Category
(See Figure 3.2 for full text of strategies)

		%							
	Total*	Try	Read	Ask	Help	Stmbl	Notice	E-mail	Other†
Totals									
all informants	60	37	26	16	9	4	3	2	7
all but E/8hr > 3.0	30	15	11	8	3	2	3	0	2
by Gender									
female	28	17	8	11	6	0	2	2	4
male	32	20	18	5	3	4	1	0	3

* total number of single Eureka slips in the category.
† several user-defined strategies, no more than 2 Eurekas using any one strategy.

Table 3.7 presents another breakdown of the Eureka data, this time by strategy used to solve the problem. The breakdown by gender suggests that female users are more likely to ask for help while males are more likely to read the manual. This data is skewed, however, by the fact that one of the high-E/8hr informants was a female who asked for help 6 times while working with a minicomputer system that had no manual, while another high-E/8hr user was a male who used the manual for 9 Eurekas, many of them while learning a system that no one else in his office was using.

3.4 Discussion

The regularities (and irregularities) in the data need to be regarded with some caution. Although the informants were generally cooperative, they clearly had somewhat different ideas of what constituted a "Eureka." The daily debriefings helped to regularize this view (this effect was weakest with the two informants who were debriefed over the phone), but there were also other factors that could have affected the number of Eurekas recorded. The informant who figured out how to get a new database program to run probably wouldn't have had time to fill in a separate slip for each fact learned. Similarly, an informant under pressure of an impending deadline might learn something and forget to record it at the time or mention it in the daily debriefing.

Nonetheless, even with these cautions in mind there remains a strong indication that most users discover things about their system only infrequently, perhaps averaging only one learning event for every eight hours of computing time. There may be exceptional users or circumstances in which this rate is somewhat higher, but even the most liberal reading of the data (using Eureka counts adjusted to reflect the multiple-fact learning by more experienced users) shows maximum average rates that are unlikely to exceed two or three facts per hour.

The data are stronger when viewed as a sampling of a large number of learning events that did occur, without regard to whether all such events were recorded. In this light they are noteworthy for the similarity of learning strategies recorded across a very wide range of situations and users. There is significant evidence that the three preferred strategies are trying things out, reading the manual, and asking for help. On the other hand, users are quite unlikely to learn things by noticing other users' activities, or stumbling onto things in the interface, or communicating via e-mail or network news. On-line help falls into a middle ground.

Going beyond the cautionary remarks at the beginning of this section, the data need to be recognized as limited in that they represent only one week's worth of observation for each of the informants. As described earlier in the chapter, an effort was made to choose an "average" week. However, to supplement this data it will be useful to consider the informants' learning strategies over a longer term. This is the topic of the next chapter.

Chapter 4

INTERVIEWS WITH DIARY STUDY PARTICIPANTS

This chapter presents data collected in structured interviews with informants after they completed their one-week diary logs. The interviews confirm the strategies for task-oriented behavior that were seen in during the log period. They also indicate that task-free exploratory behavior of computers is rare, for all types of user studied, although task-free exploration outside the domain of computers is a common recreational activity.

The diary logs reported a thin time slice of the informants' working life, covering only five days. To extend the investigation of each informant's behavior beyond that short period, another research technique was chosen: the structured interview. The questions of the interview covered essentially the same topics that had been the focus of the log period, but with some extensions to exploratory behavior outside the workplace.

4.1 Method

The fourteen informants described in the preceding chapter were interviewed at a time convenient to them, after they had completed their log. Typically, the interview took place within two or three days after the log period. During the interview, the investigator asked a series of prepared questions (Table 4.1). The interviewer would follow up the informants' initial answers on each topic by encouraging them to give examples or reasons. When necessary, the interviewer would clarify a question by giving examples of the kinds of answer that might be appropriate.

All interviews were taped and transcribed by the investigator. They ranged in length from 20 minutes to 90 minutes, and the transcriptions ranged from 1500 to 7500 words, for a total of 66,000 words. The transcriptions were examined to produce the summaries in this chapter. (The answers to the first question, on background, were described in Chapter 4.)

Table 4.1

Questions Asked in the Structured Interview
(The wording of each question varied slightly from one informant to another.)

1. Can you give me some background information about your experience with computers?
2. When you get a new piece of software, how do you learn to use it?
3. When you're using a program that you already know, and you run across something that you need to do but don't know how, how do you figure out how to do it?
4. Do you ever play around with computers, just try things out to see what they will do?
5. On the Eureka slips there is a checklist of ways you might use to learn something new. Can you give me specific examples of times you've learned things using each of those strategies?
6. Do you ever "explore" things that aren't related to computers? For example, do you go shopping when you have nothing to buy, or wander around with no fixed itinerary when travelling, or take apart things just to see how they work?
7. I'm especially interested in whether people "explore" computer systems, that is, do they try to find out what programs do even if they don't have any need for the knowledge. Do you have any further thoughts on that?

4.2 Question: Learning New Software.

After the informants described their background, the interviewer asked them, "When you get a new piece of software, how do you learn to use it?"

4.2.1 Four approaches to learning

The informants identified four main ways to initially familiarize themselves with a new piece of software:

- reading the manual, usually explicitly identified as being done in conjunction with one of the other methods (8 users:3,4,5,8,9,10,11,12)
- exploring its functionality, usually in the context of actual tasks (7 users: 4,5,6,7,10,11,14),
- working through the supplied tutorial materials (6 users: 5,6,11,12,14,16),
- having an experienced user demonstrate the package (5 users: 2,3,8,9,15),
- learning the basics initially, then learning more as the task demands (5 users: 9,10,11,12,16).

In addition to the five main strategies, a few users (2,3,9,14) had taken classes on some occasion, but this was always an exception to the way they expected to learn a program. One user (12) said that watching other users was a way to learn when no training materials or docucumentation were available.

As informants recalled examples of past learning activities, it became clear that the lines between the approaches were not clearly drawn. Availability of training materials or experienced personnel partly defined the strategies selected. Where tutorials were available, users had sometimes worked through examples as designed, other times they had used the tutorials and examples as a foundation for task-free exploration, while in still other situations they had used them as guide to a package's functionality, then moved on to real-world tasks.

4.2.2 Novice versus expert approaches

Several of the less experienced users (6,9,12) described their approach to learning a new piece of software in terms of a few past instances, which had offered different learning options. Informant 6, for example, stated that he would usually "fool around with a new program until it works," and he recalled a recent example where he didn't have a manual. But he also remembered doing a tutorial on another occasion. Informant 12 remembered learning one package by watching other users in a common computing environment, where no manual was available. The same informant noted that learning was much faster for another package, where he had been able to use a tutorial and the manuals. In general, these users did not seem to have developed a consistent approach to learning a new package, or at least none they were aware of.

The more experienced users, on the other hand, had clearly defined learning strategies, which they stated without hesitation. The informants who had worked primarily with PC or Macintosh-based software typically had a single approach to using the mixture of documentation that has become fairly standard in that arena. Informant 11 always starts with the video, if there is one, then follows the installation instructions and does the tutorial, then tries small sample tasks, then turns to larger, real tasks. Informant 16 follows the installation instructions (the "getting started" guide) and looks through the on-line tutorial, if there is one. He then begins to use the program, with the manual and detailed tutorials as fallbacks for tasks he has problems with.

Informants who had worked with a wider variety of systems, including Unix and other time-shared computing environments, selected between two distinct strategies depending on the characteristics of the package they were learning. Informant 5 specifically identified two kinds of software: totally novel packages ("out of the blue"), for which the tutorial and examples were required to reveal the software's function, and packages that were similar to known systems, such as editors, for which task-oriented exploration was the preferred learning approach. He further identified two "modes" of learning behavior: *project mode,* where getting the current project completed was paramount, and *tool mode,* where efforts were directed at learning the new tool, typically by redoing a task that had already been completed with another tool. "Before I sit down at the terminal I know which mode I'm in," he said.

For informant 10, only simple software was worth a quick exploratory foray, while large, complex packages demanded an in-depth understanding of the manual and the program's philosophy. He felt large systems were not worth investigating without a real-world task to guide and justify the learning effort. For these packages, he identified a sequence of investigatory activities, beginning with the manual, proceeding to breadth-first, task-free investigation of menus and controls, and finally moving into depth-first, task-oriented work. Informant 4 made a similar distinction between simple and complex packages, stating that reading and understanding, including an understanding of the source code, was preferred to exploration for major software applications where the code was available.

4.2.3 Task-oriented versus task-free learning

Of the seven users who identified exploration as one method of learning a package, six explained (some in response to the interviewer's follow-up question) that they performed this exploration in the context of tasks. The seventh (14) did not clearly respond to the follow-up question. One user (10) performed both task-oriented and task-free exploration, as described above.

For most users, the most effective approach was to use their own tasks. Informant 5 used his own tasks because in that "demand-driven" mode he would not waste time learning what wasn't needed. Informant 11 would sometimes begin exploration with simple, trial tasks, but would soon progress to larger, real tasks. Informant 16 would look at example tasks provided in the tutorials because they demonstrated the software's functionality, but he postponed learning until a real need arose. "I just keep them in mind if I want to do something like that," he explained, "because I know the tutorial would walk you through those examples," Only Informant 4 claimed a preference for the sample tasks provided with the system, since the data for the examples were already entered.

4.2.4 Making use of manuals

Although eight informants mentioned using manuals to learn new software, only three (4,5,10) described situations in which they would start by reading the manual in depth. These users were all highly experienced computer scientists, and the occasions when they would read the manual typically involved learning a complex new programming environment (although Informant 10 would prefer to read the manual for in-depth understanding before tackling any large-scale application).

For the other informants, the manual was a supplement or a support to other strategies. Informant 11 would scan through the manual, but only after exploring the program for a while: "I take the manual someplace away from the computer... just to browse through it and look for features that might be interesting or useful later on." Informant 8 liked to get an overview of the program through the manual, then have an experienced user demonstrate how to get the package running.

For the majority of users, the manual was most valuable as a fallback when they ran into problems, the situated investigated in Questions 2 and 4. Informant 16 expressed the consensus opinion of the value of manuals for initially learning about a program: "Reading the User's Guide is, to me, useless. I only use the User's Guide when I want to do something specific."

4.2.5 Time constraints

Many of the users explicitly identified time as a constraint on their learning activities. Informant 2 didn't like "reading through all that stuff [the manuals] and trying to find the answer to my problems." He preferred a personal demonstration. As noted above, Informant 16 never read the manual until he had a specific problem to resolve, and he initially looked at the examples only to see what could be done. However, he liked the on-line tutorials, "because they're short, and they show you a lot of things right up front."

Informant 3 volunteered his opinion of unstructured exploration: "It's not that I don't like to do that, but I feel like if I'm working, it's not a good use of time." Informant 7 described an extended exploratory session driven in part by curiosity but also guided by a "cost/benefit analysis." His initial impression of the software was negative because of obvious problems, but he decided to explore further to see if it contained features that might make the earlier problems worth overcoming.

Informant 5, who had mentioned the efficiency of the "demand-driven" approach to exploration, also described how he learned systems "incrementally": He would discover a problem, try to solve unsuccessfully to solve it for a while, then continue with his work. When the same problem came up again on another day, he would take a few minutes to try a different solution, but would give up again if that failed. "You know, I want to get this thing edited," he said, referring to the word processing package he was currently learning "and if I can't find a feature fast, I'm just going to forget it."

4.2.6 *Summary*

When learning a new piece of software, inexperienced users are likely to select whatever method is available to them. More experienced users, however, select the learning strategy that they believe will help them acquire the skills they need with the least possible investment of time. With relatively simple or standard packages and experienced users, exploration in the context of the user's immediate task may be an effective first strategy. With complex, novel packages, however, experienced users prepare themselves for task-oriented exploration by reading the manual, working through on-line tutorials where available, and availing themselves of their colleagues' expertise through brief demonstrations.

4.3 Question: Resolving Problems.

For many of the informants, there was no clear distinction between learning a new piece of softare and resolving problems with a software package they had been using. This was the topic of the next question: "If you're working with a program that you already know how to use, and you run up against something that you don't know how to do, how do you figure out how to do it?"

Three strategies for resolving problems dominated the informant's answers:

- trying things out (exploration) was clearly identified as a first strategy for 9 users (3,4,5,6,11,12,14,15,16). An additional 2 users (3,8) described their first action as a combination of looking in the manual and trying things out.
- looking in the printed manual was a first strategy for 3 users (2,9,10) and a second strategy for 7 (2,4,6,11,12,14,16).
- asking for help was a first strategy for 2 users (3,7), a second strategy for 5 (2,6,8,9,15), and the third strategy for 4 (4,10,11,14).

Additional strategies identified were working around the problem (informants 4, 5, 10, and 12, the first three of whom would sometimes use programming for the work-around -- although informant 4 also praised the efficacy of white-out), using on-line help (informants 5, 7, and 11), and looking at the source code (a first strategy in complex systems for informant 4, and a second strategy, after asking colleagues, for informant 11).

Note that some informants identified alternate "first" and "second" strategies, depending on the problem, time and resource availability, and in two cases (2,3), their mood at the time the problem arose. Informant 2: "It depends on my mood [laughs], if I want to talk to somebody or if I want to read it."

Many informants distinguished between things they would immediately ask system support personnel to handle and things they would try to handle on their own. Hardware problems were almost always referred to systems support. "That's sort of their job," explained informant 12.

4.4 Question: Task-Free Exploration of Computer Systems

All but one of the informants answered that they did little or no exploration of computer systems except for the purpose of performing current or impending tasks. The one informant (11) who identified this as a common activity had, according to the diary log, spent one third of his computer time exploring new software. Other examples he gave of exploratory behavior were changing the appearance of his on-screen workspace, "about once a day," and producing graphs or charts for course materials that were unnecessarily decorated. "I really don't leather-look bar charts," he admitted. "And I don't really need a sophisticated, vector-based graphics program." He specifically stated that he was a little embarrased about this behavior, because he knew it wasn't productive.

One other informant (9), an undergraduate, said he liked to do task-free exploration, but didn't have much opportunity. "Almost every program, I try to find something fun about it," he explained, giving the example of learning to use the sound-recording capability of his department's Macintosh to record alarm messages in his own voice and the voices of his co-workers.

The remainder of the informants answered either that they never or almost never did that kind of thing, at least not in their current sitution. "Only once have I ever done that in my life," said Informant 5, and then gave the details of his experience with a Mandelbrot set program. Four of the informants (4,8,10,14) recalled doing more task-free exploration in the past.

Essentially there were two reasons expressed for not exploring computer systems. The most common reason was time constraints. Seven of the informants (3,4,6,7,8,14,15) specifically said they didn't have the time or they felt that exploration was not productive, at least not in their current job.

The second clearly expressed reason stated for avoiding task-free behavior was that it was simply not interesting. "I use [computers] only as an instrument, and I don't see them as something fun," explained Informant 15. A similar attitude was expressed by Informant 8: "I don't explore the actual computer all that much. Of course, the computer's a tool." And Informant 12, while admitting to have tried a few adventure games, ranked the computer low on the scale of interesting activities: "For leisure I like to run around outside or watch TV. I wouldn't normally come up with the idea of switching on the computer to entertain myself."

4.5 Question: Eureka Strategies

The informants had stated their preferred strategies for resolving problems, and these preferences were largely validated by the Eureka counts. It will be useful to consider the the reasoning behind these preferences. Were the preferred strategies simply the only ones the informants had ever tried? Or, more likely with the experienced users, had they tried many strategies but found the preferred ones most efficient?

To investigate these questions, and to provide the informants with a different index into their exploratory experience, the interviewer asked: "You've been using these Eureka slips in the diary study. Can you remember specific occasions in the past when you learned something through each of the methods listed on the slip?" For strategies discussed with users earlier in the interview, the answers were usually short, but for other strategies the question provided a springboard into discussion of the strengths and weaknesses of the approach.

The informants' answers indicated that most of them had tried the strategies available to them, but the ease with which they recalled specific instances echoed the strategic preferences they had stated in Question 1. (In the interviews, this question was asked after Question 4, Task-Free Exploration, so informants were not so likely to immediately recall what they had just said in Question 1.) The results of the question are summarized in Table 4.2.

4.5.1 Strategy 1: Read the paper manual

All informants recalled instances of solving problems by reading the software's manual. Only one user, who had stated a strong preference for exploration and asking for help, hesitated briefly before answering affirmatively and providing the example. For many users, the most useful manual was a commercial "how-to-use-Program-X" sort of manual, written by someone other than the software manufacturer. In addition, users of networked systems often maintained written notebooks of procedures learned in training courses or from interactions with system support personnel.

Table 4.2.

Answers to Question 4, "Can you recall examples of learning things using each of the categories in the Eureka report?"

Strategy	**Used**		**Not Used**	
Read paper manual	all			
Use on-line help or man	*yes, praise or* *no comment* 2,5,9,14	*yes, but* *problems* 4,7,8,10,11,12,1 6	*tried, doesn't* *work* 3,15	*no, no comment* 6
Tried things until it worked	*yes, de novo* all but three	*yes, with* *manual* 8,9,10		
Stumbled onto by accident	*yes, in* *interface* 9,11,12	*yes, in manual* 4,5,8,15	*maybe, can't* *recall* 2,3,6,7,10,14,16	
Asked in person or by phone	*colleagues or* *sys. support* all			
Sent e-mail	*unqualified yes* 5	*yes w/* *reservations* 4,8,11,12	*no, too slow* 2,3,12	*no, no comment* 6,7,9,14,15
Posted to network News			*no, for reasons* 2,3,4,7	*no, no comment* 6,8,10,12,15*
Noticed someone else	*yes,* *serendipitous* 2,4,5,7,9,12,14,1 5,16		*only in training* *or demo* 6,8	*no, no comment* *or can't recall* 3,11
Other	*yes (usr grp,* *video, class)* 4,11,16			*no, can't think* *of any* 6,9,15**

* Informants 5,9,11,16 didn't answer the "net news" question
** Informants 2,3,5,7,8,10,11,14 didn't answer the "other" question.

4.5.2 Strategy 2: Used on-line "Help" or "Man"

All but one of the informants had tried on-line help or man, but for the majority of them it was not highly regarded. Three users gave an example of usage without any qualification (Informant 2 in Unix, 9 in Word Perfect, and 14 in DOS), and one experienced user (Informant 5, user of DOS, Macintosh, and Unix) described on-line help as his first fallback after trying things failed. The remaining comments were strongly negative or qualified.

Two users, both heavy Macintosh users, had tried on-line help and decided it was useless. "It never, ever, ever works for me," stated Informant 15, explaining that he either didn't understand the help message or else it clearly didn't answer his question.

Informant 3 also complained that the messages were difficult to understand, and that they gave too much useless information: "I mean, you ask it for help and it tells you everything in the universe."

The remaining users recalled instances of finding things using on-line help, but qualified their opinion of the approach. Informant 11 would use on-line help on DOS systems, but only if the manual wasn't readily available. Five informants would use on-line help for some systems (VMS, Unix, Macintosh, SAS), but not for others (Unix, Macintosh, DOS).

Comments on Unix man, even by informants who used it, were negative (except for Informant 2, who had no positive or negative comments): Informant 4 uses man but finds it "generally useless." Informant 7 said it "doesn't usually amount to much." Informant 8 would "rarely use the man stuff" and gave an example of how it failed to resolve a problem. Informant 10 thought man was "the worst thing in the world."

Informants 4, 8, and 12 would use on-line help for command-oriented systems (Unix, VMS, SAS) but not for the software they had used on the Macintosh. Informant 4 thought the Mac help was "crap," especially the balloon help: "I haven't found any of those messages useful." Informant 8 would rather use the paper manuals, while Informant 12 preferred to explore and complained that Microsoft Word's help kept popping up when it wasn't wanted. On the other hand, Informants 10 and 16 would use on-line help with the Macintosh but not with command-line systems (Unix and DOS, respectively).

It's interesting to note that the three informants (5,10,16) who liked Macintosh on-line help are users with extensive experience, including strong backgrounds in command-line systems (Unix and DOS). On the other hand, three of the five informants who did not like Macintosh on-line help (3,12,15) had limited experience with command-line systems. Informants 8 and 4 were exceptions to this pattern, having strong command-line experience but disliking Macintosh help.

4.5.3 Strategy 3: Trial and error

All informants recalled instances of trying different things until they had resolved a problem. Many of the examples had been given in answer to an earlier question or had arisen during the diary study, and there wasn't a lot of further discussion. Three of the informants (8,9,14) noted that they often used trial and error to disambiguate information from the manual. (Informants 8 and 3 had made the same comment in response to question 1).

One of the highly experienced informants (10) noted that this approach was common in programming (it's one definition of "hacking"). He made the distinction between a "quick and dirty" approach and doing a job well. He associated trying things out with the quick-and-dirty approach, and said he would try to read the manual and understand the software if he wanted to learn it well. He had earlier provided details on his learning experiences with Microsoft Word's "styles" feature, where he described how the trial-and-error approach had led him to use the program in an inefficient manner.

Another informant (16) noted that he also used trial and error for household appliances, such as a microwave oven. He added that when the approach failed, he usually gave up, and gave examples of his failure to set the time on a car clock or program a VCR.

4.5.4 Strategy 4: Stumbled onto by accident

The phrase, "Stumbled onto it by accident," was intended to cover unplanned learning instances, such as intending to type tab to move the cursor forward, but accidentally typing shift-tab and discovering that it moved the cursor back. However, several informants gave examples of other unstructured learning strategies, such as trial-and-error or noticing someone else, both of which fall within a reasonable interpretation of the phrase.

Once they understood the question, informants generally had trouble recalling instances. The only situation in which subjects easily recalled stumbling across new features was when reading manuals, where Informants 4, 5, 8, and 15 had noticed things they weren't looking for. Every one of the other informants said something to the effect of, "I'm sure that must have happened, but I don't know if I can think of an example." Only two informants actually recalled stumbling onto a new feature in an interface: Informant 11 discovered a table-formatting option that was in an inappropriate dialog box, and Informant 12 discovered Microsoft Word's "drag-and-drop" feature. Finally, Informant 9 recalled discovering how *not* to do something in a spreadsheet -- that is, he made an error and the program caught it.

In summary, the question might have been better worded, and it is possible that the informants simply hadn't indexed their experiences for retrieval through the cue provided. But it seems most likely that stumbling across new features in an interface -- and remembering how to use them -- is a rare occurrence.

4.5.5 Strategy 5: Asked someone (in person or by phone)

All informants easily recalled asking for help on computer software problems. The interviews provided weak evidence of a correlation between availability of software help and willingness to ask. The informant most likely to ask for help (15, who said he would "always!" ask for help) was a faculty member in a department with responsive system-support personnel. Three of the four informants who pointed out problems with asking for help (4,11,14) were in situations where support was not so readily available. But no clear measure of willingness to ask or availability could be derived from the interview data.

Several factors were raised that biased users against asking for help. Two informants (11, 14) worked in situations where they were usually the most experienced user of their software, so they could only get help from the phone-support lines, which they both did. Another informant (4), who reported frequently calling a consultant for help, gave the opinion that some expert users were annoyed by the frequent questions from other users. Still another constraint was time, already mentioned in connection with Question 1. Informant 12 stated that he would ask for help if someone was in the room or in a nearby office, but if no one was readily available he would work around the problem, leaving the question to be resolved later when a colleague was available.

An additional factor, pride in one's ability to solve problems, was hinted at in some of the interviews, but the bottom-line analysis indicates that the stereotype of the lone computer scientist may be a false one. Informant 5, a computer science graduate student, summarized the case nicely: "I'm not an asker," he said. "Although, this Ph.D. program has changed that. I ask more questions than I used to. Something about graduating that appeals to me."

4.5.6 Strategy 6: Sent e-mail or posted news request for help

Neither e-mail nor network news programs were widely used by the informants as a problem-solving resource. Only five of the users recalled sending e-mail requests for help. One informant (16) preferred e-mail over phone conversations because it allowed a more detailed communication, but in the end he would select whichever media provided the fastest response. Informant 5 was noncommittal, giving an example of a successful e-mail request to a software author. Informant 11 used e-mail "very seldom," and Informant 8 considered it only an alternative to a phone call. Informant 4 complained that " e-mail doesn't work too well for me, because either I don't explain things too well or people don't pay attention." He added that he would only send e-mail to systems personnel, since sending an e-mail request to a colleague would be insulting.

Three of the users who said they would never use e-mail gave time constraints as the reason. "If a problem comes up, you don't want to wait on it," explained Informant 2. The other users simply stated that they didn't use it or preferred phone calls.

None of the informants recalled posting to network News with a software problem. Four users gave no specific answer to the question. Eight more simply said "no." Informants 3 and 12 said they would never ask a question on News because it was too humiliating.

Informant 3 also thought News was too slow. "I don't want to come back two hours later and get 20 replies." Informant 7's expectations were even lower: "If you post something saying, does anybody know how to do this? Then you get back twelve replies from people saying, yeah, when you find out, tell me. Great. That's really useful."

4.5.7 Strategy 7: Noticed someone else doing it

None of the informants had a quick answer to this question, although 11 of them eventually recalled examples of some sort. Six of the examples involved noticing a specific command given by another user on a computer (2,4,12,14,16), although the interviews did not always reveal whether the informants had noticed the command itself or merely noticed its effect and then asked how to achieve it. Another three (5,7,9,15) clearly recalled noticing the effects, which included a different text previewer, a text format on an overhead slide, a sound output, and a way of combining two forms of data. The remaining two occasions were in the contexts of demonstrations or training sessions, which was not the intended meaning of the question.

Several interesting factors were noted by informants as they considered this question. Informant 2 pointed out that this kind of learning event happened more frequently when his workgroup had all just started on a new system. Informant 11, who couldn't recall an example, is a faculty member who explained that he almost never worked in an environment where other people were using computers. Informant 15 stated a strong preference for asking for help, and his example of noticing involved noticing that a colleague could do something with the computer, then going back to the colleague at a later date and asking how to do it.

4.5.8 Strategy 8: Other

One informant (4) mentioned user groups as a place to learn things, another (12) suggested videos. Neither recalled specific instances. A third informant (16) mentioned classes, and several other informants had recalled attending classes when they described their background. Most of the informants, who were reading the categories off a Eureka slip as they gave their answers, simply skipped this category.

4.6 Question: Task-Free Exploration Outside of Computers

Question 4 established that task-free exploration of computer systems was uncommon for the users studied. It is interesting to compare the informants' behavior in the domain of computers to their behavior in other domains. It might be the case that some users are especially inclined to exploratory activities in any domain, an inclination which might show up in their computer activities. Conversely, if the users did not explore in any domain, then their failure to explore computers may reflect a more fundamental preference or limitation.

To investigate these issues, informants were asked to recall exploratory activities in non-computer domains. Because the words "exploratory activities" would have little meaning to the informants, the question was supplemented by examples: shopping with nothing to buy, or travelling without a detailed itinerary.

All fourteen of the informants readily recalled examples of unplanned, goal-free activities, and all of the activities were recreational. (This may have reflected a bias in the examples given as part of the question.) Travel was the most common domain for exploration. Other domains included shopping and hobbies, such as gardening.

Eleven of the informants reported exploratory activities as part of their travel experiences. Within this group, however, there was a wide range of exploratory freedom. One informant described several weeks of bicycling through Europe, during which only the trip's endpoint and one intermediate stopover were preplanned. Each day's activity included a check of the map to see what towns were within biking distance as possible places for the next night's lodging. Another informant described a superficially similar trip: bicycling through Europe for several weeks. However, for this trip all the details had been planned on a spreadsheet, down to the individual hotels and restaurants. Only the activities to fill a small amount of spare time were left unplanned and open to exploration.

4.7 Question: Influence of Exploratory Activities

It had been suggested that exploration, especially in adults, might be rare, but that it might be the source of significant learning events that led to major changes in a person's life. A history of fascination with how machines work might lead someone to study mechanical engineering, for example. To investigate this possibility, informants were asked whether their exploratory activities had ever led to discoveries that significantly influenced their lives.

None of the informants could think of a significant example of exploratory activities that had any significant influence. This negative result is in part related to the answers given to the previous question, about exploratory activities. The activities listed most frequently involved travel and shopping, and informants were hard-pressed to imagine how these activities might have a long-lasting influence.

As a follow-up question, informants were asked how they came to select their current career and how they had made other major decisions leading up to what they were doing at the time of the interview, such as deciding where to go to college or where to live. The answers to these questions provided a rich set of examples of serendipity. Most of the informants were able to recall some unplanned occurrence that had aroused their interest in a topic, or supplied information about a possible choice of university or place to live. But none of these serendipitous occurrences had anything to do with exploratory activities.

4.8 Discussion

Overall, the data collected in the interviews validated and helped to explain the behavior recorded in the daily logs. Most importantly for our research into exploratory learning, the interviews emphasized that users engage in task-free exploratory learning only on very rare occasions. However, they do use exploration as one means to resolve task-oriented difficulties, typically combining this approach with looking up things in manuals and asking for help from other users or system support personnel.

Chapter 5

COGNITIVE MODELS OF EXPLORATION

This chapter describes a series of low-level models of exploratory behavior, in three different cognitive architectures. The modelling efforts provide a detailed analysis of the knowledge used in problem-oriented exploration. Common features of the models suggest important cognitive strategies and problem areas.

Chapter 3 described a formal model of exploratory behavior, which revealed some of the constraints of the task of exploration. Chapters 4 and 5 investigated how users respond to those constraints, yielding the important conclusion that most exploratory behavior occurs in the context of specific tasks that arise as part of the user's work. This chapter describes several cognitive models of task-oriented exploratory behavior, at a grain size that is much finer than the general observations derived from the diaries and interviews. In Chapter 6, empirical studies of users in the laboratory will focus on a similar level of behavior.

With the exceptions of the Kitajima-Polson model (Section 5.2) and Richard Young's Soar model (Section 5.5), all models described in this chapter were implemented by the author and reflect ongoing discussions with Clayton Lewis and Peter Polson.

5.1 Background

The models described in this chapter all reflect a fundamental understanding of goal-oriented problem solving that is founded in Newell and Simon's theories cognition and artificial intelligence. Models in this family were initially developed in the context of well-defined tasks, such as games and mathematical puzzles (Newell, Shaw, and Simon, 1958; Newell and Simon, 1972; Simon and Greeno, 1988). Newell and his associates have further refined the goal-oriented problem solving engine in the SOAR computer model, and shown that the model predicts established empirical results, not only in problem solving but in learning, language, and other areas (Laird, Newell, and Rosenbloom, 1987; Newell, 1990; Rosenbloom, Laird, and Newell, 1991). Simon and other researchers have suggested that the same fundamental approach can also describe ill-structured tasks (Simon, 1973), such as scientific discovery (Klahr and Dunbar, 1988) and software design (Polson, Atwood, Jeffries, and Turner, 1981).

The branch of this work underlying the current research is the CE+ theory of learning by exploration, proposed by Polson and Lewis (1990). This theory merges ideas described in cognitive complexity theory (CCT) and EXPL. CCT is a production-system model for formally representing a user's knowledge of an interface (Bovair et al., 1990; Kieras and Polson, 1985), based on the GOMS model of human-computer interaction (Card, Moran, and Newell, 1983). To this basic framework, EXPL (Lewis, 1988) contributes heuristics for learning new functions.

The high-level description provided by the CE+ theory was further refined by considering its instantiation within the framework of Kintsch's construction-integration model (Kintsch, 1988; Mannes and Kintsch, 1991). The construction-integration model describes the processes by which users integrate a representation of text or other perceptual input with background knowledge to construct a representation that will enable them to perform a task.

5.1.1 Goals and Actions

In outline, the CE+ theory, understood in terms of the construction-integration model, describes the cognitive component of exploratory behavior as follows: An initial goal structure is constructed from a description of the user's task. Goal structures in the theory are similar to the goal hierarchies postulated by the GOMS model (Card, Moran, and Newell, 1983; Bovair, Kieras, and Polson, 1990; Kieras, 1988), with a top goal representing the overall task, intermediate level goals defining a task decomposition, and lowest-level goals describing individual actions.

Goals are represented by propositions. They are linked to other goals, to propositions representing background knowledge, to propositions representing objects seen in the environment, and to actions. Activation flows from the top goal along these links to the representations of actions. When an action becomes sufficiently activated, it is executed. Any response by the system is observed and interpreted, causing accomplished goals to be deactivated and new propositions to be built. These propositions represent new goals and changes in the environment caused by the last action. The new propositions are linked into the existing network of propositions. Activation now spreads through this new network. The next action occurs when some action becomes sufficiently active, and a new cycle begins.

As just sketched, actions are executed when sufficient activation reaches them. For this to happen there must be a path of associative connections between a user's goal and the representation of the action. One situation in which such a path exists is the label-following strategy employed by naive users (Engelbeck, 1986; Polson and Lewis, 1990). Here an action, such as pressing a button, is chosen because there is a label associated with the action that shares terms with an active user goal. For example, a user with the goal of turning a system off would be expected to press a button labelled 'off'. In this case, the associative path between goal and action is composed of at least four linked propositions: the representations of the user's currently active goal, the button label, the button description including its location, and the action of pressing the button. Activation spreads across these links, and if it reaches a sufficient level, the button will be pressed.

Not all actions must be linked to goals in as simple and direct a way as required by the label-following heuristic. However, some added knowledge must be assumed if the user is expected to link an action to a goal without benefit of a label or analogous cue, or expected to link a goal with a label that does not share terms with the goal. For example, if a button is labelled '1/0' rather than 'off', a successful user must know what the label means. Careful attention to questions of existing knowledge will allow the model to reflect the behavior of expert system users faced with new applications or functionality, as well as novice users who are approaching the system for the first time.

5.1.2 Extending the Theory

The research described in this chapter addresses two fundamental limitations of the existing theory and models. As described, the CE+ theory is essentially a problem-solving engine. Like most Newell-and-Simon style models, it relies heavily on a hierarchy of well-defined goals. But as exploration has been defined in this research (see Chapter 1), the explorer's immediate goals are often unclear, and there is seldom any preestablished subgoal hierarchy, or "plan." While the formal analysis in Chapter 2 and the field studies in Chapters 3 and 4 suggested that truly goal-free behavior was rare, they also suggested that users were able to work with ill-defined goal structures, which they refined based on the ongoing interactions between the existing goals and the interface.

This opportunistic behavior is a clear instance of the "situated cognition" that traditional problem-solving models have, according to many critics, failed to represent (Neisser, 1967; Suchman, 1987; Lave, 1988; see Brooks, 1991, for related AI work). Situated behavior is characterized by its responsiveness to a changing environment and by its lack of reliance on preconstructed plans. The theory needed to describe explicitly how this situation-based reasoning could occur.

The second fundamental weakness of the theory was that it did not provide a story of learning. Again, the theory presented a problem-solving engine. Certainly some of the correct actions uncovered by the problem solving would be remembered, but which ones? Again, the theory needed to be extended clarify this issue.

5.1.3 *Models Described in this Chapter*

The rest of this chapter presents a series of models intended to address the issues just described. Several early models are described only briefly, as a record of the historical course of the research. The most advanced model, ACT-R version 4, achieves the basic goals set for the modelling effort. That model is described in considerable detail and compared to an alternative approach implemented in Soar. Partial code and traces for models developed by the author are shown in an appendix of the author's dissertation, but are not included in this publication.

5.2 A Construction-Integration Model of Expert Behavior

The cognitive modelling work that laid the immediate foundation for the current research was a comprehensive effort by Kitajima and Polson to program and investigate the behavior of a model of expert behavior with a well-defined task in a display-based computing environment (Kitajima & Polson, 1992). This work was done in the context of Kintsch's construction-integration theory (Kintsch, 1988) with a system implementation developed by Mannes and Roushey (Mannes & Kintsch, 1991). The task was to start the Cricket Graph application on an Apple Macintosh, to create a graph using data in a supplied file, and to edit certain features of the graph, such as title size and legend placement.

The Kitajima-Polson model has a structure that echoes the yoked state space hypothesis of Payne, Squibb, and Howes (1990). It provides a principled explanation of errors observed in expert behavior, and it describes ways in which the experts' subgoals and the computer display interact to locate an attentional focus. However, it does not describe the exploratory behavior of a first-time user of the Cricket Graph package. This was the goal of the modelling efforts performed in this dissertation.

5.3 The Novice Construction-Integration Model

The initial model of novice exploratory behavior was implemented under Kintsch's construction-integration theory. The same Cricket Graph environment and task were assumed as in the Kitajima-Polson model. However, the new model described a user whose experience covered basic Macintosh applications, but not Cricket Graph. The computational substrate was the system developed by Mannes and Roushey, supplemented by the"wrapper" code developed by Kitajima. Kitajima's code made it simple to develop the model iteratively, with repeated testing.

The Kitajima-Polson model described the behavior of an expert user. It relied heavily on knowledge that reflected complete and complex representations of the screen, the Macintosh control conventions, and the task. To produce a model of novice behavior, the Kitajima-Polson model's knowledge structures were completely eliminated. They were replaced with the simplest possible representation of the screen state, along with simplified plan elements, including a "move-to" plan, a "double-click-on" plan, a "release-on" plan, and a few others.

A very significant change from the Kitajima-Polson model was the use of a single unstructured goal for most of the task. The goal was nothing more than an unordered sequence of keywords, which produced "label-following" behavior by overlapping with screen features when appropriate.

There were problems with the model. First, the unstructured goal failed to take the model through the axis-selection dialog box. To achieve this final step, the goal had to include two minimal subgoals, each associating the name of an axis with its data. Further, only one of these subgoals could be strongly active at any given time. This was accomplished by establishing a focus of attention on the screen, then using the activation from that screen element (e.g., the list label "x-axis") to activate the appropriate subgoal (e.g., "x-axis: observed"). While this was an effective solution, it was not an elegantly simple one, and it required knowledge structures and manipulations of state that were not predicted by the basic construction-integration model.

A second, deeper problem involved learning, both long-term and in support of the immediate task. The construction-integration model, in the implementation we were using, did not support learning. But without some kind of learning, the model could be doomed to repeatedly perform the same actions, since it could not remember that that part of its goal had already been satisfied. When a screen update prevented such repetition, this was not a problem. But in the case of the axis-selection dialog box, both axis lists remained visible, and there was no elegant way to ensure that the model would click the OK button after the axis selections had been made.

5.4 The ACT-R Models

To overcome the shortcomings of the novice construction-integration model, and to investigate similar issues under somewhat different assumptions, we decided to move our modelling efforts to a different cognitive architecture. We chose Anderson's ACT-R (Anderson, 1993).

The ACT-R architecture is the most recent of a series of ACT theories of cognition (Anderson, 1983). It is the first version of ACT for which an implementation has been generally available outside of Anderson's laboratory. Like all ACT versions, ACT-R incorporates two forms of long-term memory: declarative memory for facts and production memory (rules) for cognitive skills. A small portion of long-term declarative memory is active at any one time, forming the system's Working Memory. ACT-R is distinguished from earlier versions of ACT by its rational analysis component, which replaces traditional conflict-resolution rules. The rational analysis mechanism considers cost and predicted success of competing productions and decides which to fire. There is a single method for learning new productions in ACT-R: analogy.

The implementation includes no clearly defined mechanism for long-term declarative learning or for forming declarative categories. This is significant because much of the power of the analogy mechanism derives from its ability to work with identities and differences between slot fillers of two "Working Memory Elements" (WMEs) that fall into the same declarative category. Every WME must be identified as belonging to a predefined declarative category, termed a "WME Type," and the WME Type defines exactly the slots that WMEs of that type can have filled. These slots, in turn, are examined by the analogy mechanism. In Anderson's addition model, for example, there is a WME Type for addition facts, which has attribute slots labeled "addend1," "addend2," and "sum." An individual addition fact can then be represented as: (fact-1 is-a addition-fact, addend1 5, addend2 4, sum 9). A major goal in the modelling work was to reduce the dependency on categorical structure, so the model could learn as much as possible from scratch.

Several models of the Cricket-Graph task were implemented in ACT-R. The first efforts are described only briefly, to highlight certain problems with the system, as well as insights into the cognitive demands of the task. The final model is described in considerable detail.

5.4.1 Version 1

The first version of the model reflected a traditional, goal-oriented AI approach. The model was provided with a list of things that needed to be accomplished, and it worked its way through that list. Knowledge of the current world state was maintained using categorical structures were simple, but were explicitly designed to represent exactly the information needed to support analogy. The model was able to learn general rules by analogy to previous instances of similar problem-solving episodes. For example, given the goal of starting Cricket Graph, and recalling a previous episode in which double-clicking the Microsoft Word icon started that program, the model would infer that double-clicking any application's icon would start that application.

The critical failure of this model was its inability to describe the "situatedness" of behavior that seems to correctly reflect a novice user's behavior. The model relied on a list of higher-level goals, and within that list it pushed and popped subgoals onto the goal stack, an activity that ACT-R explicitly supports but that clearly locked the model into behavioral states that could not be broken out of, even if the screen were to change in an unexpected manner. A second failure of this model was the specificity of its categorical structures.

5.4.2 *Version 2*

In the next version of the model, the categorical structures were redesigned to provide the absolute minimum of information. Using a convention suggested by Clayton Lewis, the structure of each element in working memory was essentially reduced to a simple subject-verb-object triple, termed a "prop" (proposition). Additional categorical structures ("Working Memory Types" in ACT-R terminology) were provided for actions and effects, which ACT-R required for analogies to form, and for objects, which were simply anchor nodes referred to by the props.

This model performed the basic Cricket-Graph sequence required to create the default graph: double-click on the data icon, pull down the graph menu, release on "Line Graph," associate data with the x and y axis, and click on "OK" in the dialog box. It relied on a large network of basic Macintosh facts, such as the fact that icons can be double-clicked. In addition to its simplified categorical structure, it also satisfied the constraint of situatedness: the goal stack was almost never used, except to maintain the single, top-level goal: "figure out what to do." The model performed the task through "label following," comparing the current screen state to an unordered list of instructions describing the task. Instructions and objects were marked "processed" in memory to prevent repetitive actions.

The model's shortcoming, however, was that it did not learn. The declarative knowledge now met the constraint of being categorically simple and hence learnable in principle, but this simplified structure no longer supported analogical formation of new productions.

The model also had potential problems related to its strategy of "dumb" label following. These problems did not surface in the early stages of the Cricket Graph task, but our examination of several other graphing programs, part of a "bestiary" collected by Peter Polson, suggested difficulties that could arise. The problem was this: The model looked for overlaps between instructions and screen objects, then blindly acted on the screen objects when overlaps were found. But what if an object appeared on the screen that matched an instruction which had already been used to select a different object? If the instruction had been satisfied, the model would not act on the overlap again. But if the instruction were only potentially satisfied, as it would be if a dialog box had been partially filled in, the model would act again -- and again, and again, and again, if the object continued to reappear. This problem, along with problems in deciding when to click the "OK" button in a dialog box, suggested that blind label following was not a sufficiently sophisticated strategy. What was needed was the ability to predict an action's result and decide on its appropriateness.

5.4.3 *Version 3*

The next version of the model incorporated the look-ahead needed to supplement the label-following strategy. This involved a major revision of the knowledge base and the model's control strategies. Where the previous model had simple looked for label-instruction overlap and acted, the new model looked for label-instruction overlap, envisioned the result of acting, and then acted if that result was appropriate. Two tests of appropriateness were made: (1) did the envisioned effect match the instruction? and (2) had the effect not yet been achieved?

Not only was the knowledge base heavily revised to support this more sophisticated strategy, but the knowledge was significantly shifted from declarative to production form. Where the earlier model had contained declarative knowledge that icons were double-clickable, this model had a production which proposed double-clicking icons, with an envisioned result being produced as part of the proposal.

This sophisticated version of the model was able again to complete the entire first part of the Cricket Graph task, from start-up through double-clicking the "OK" button. Furthermore, it did so in a manner that would have supported other interfaces that provided fewer guiding constraints than Cricket Graph. However, the model still did not learn.

5.4.4 *Version 4: Overview*

Version 4 of the ACT-R model finally achieved our basic goal of modelling plan-free exploration with learning. This subsection provides an overview of that model in its current form. Details of the ACT-R implementation are provided in the next subsection. The overview also applies to the Soar model, described briefly in Section 6.3.

The task is represented in the model as instructions that have been acquired by some parsing process that we do not describe. The model knows how to start at least one other program, Microsoft Word, by double-clicking on its icon, but it has no general rule for starting Mac programs. Knowledge in the model is contained partially in declarative form and partially in rules ("productions" or "chunks").

The reasoning process proceeds as follows. The model sets the goal of starting Cricket Graph, but no rule fires to achieve that goal. An impasse occurs. In response to the impasse, the model attempts to use knowledge of similar situations to solve the current problem.

The model recognizes that starting Microsoft Word is similar to starting Cricket Graph. It considers the similarities and the differences between its experience with Word and its current Cricket Graph task, and it concludes that double-clicking the Cricket Graph icon will achieve its current goal. It takes that action and recognizes that the goal has been satisfied. To make the results of its reasoning available for future situations, it forms a new rule of the form: "If you want to start Cricket Graph, then double-click on the Cricket Graph icon."

The analogical reasoning process, summarized in the previous paragraph, can be broken down into several steps, similar to those identified in Holland, Holyoak, Nisbett, and Thagard's (1986) general model of analogy and to the more abstract process of metaphor described by Carroll, Mack, and Kellogg (1988).

1. Recall a previously accomplished *example task* that is potentially analogous to the *current task* (also called the analogy's source and the target, respectively).
2. Identify a *mapping* between the conceptual structures of the example task and the current task.
3. Describe a *new action* that will accomplish the current task. Do this by applying the mapping to the *example action* that achieved the example task.
4. Test the analogy by trying the new action.
5. Learn a new rule (production) for this situation, so the cognitive overhead of performing analogy can be avoided if an identical or similar situation arises again.

The next subsections describe each of these steps in detail.

1. *Recalling Candidates for Analogy*

To begin the analogy process, the model must retrieve knowledge about past interactions (example tasks) that might suggest a method for completing the current task. Empirical research suggests that the retrieval process is highly dependent on surface similarity between the current task and the example (Chi, Feltovich, & Glaser, 1981; Gick & Holyoak, 1983; Holyoak & Koh, 1987; Ross, 1984). In the domain of computer interfaces, for example, Ross (1984) taught users two equally successful methods for a word-processing task, such as append and insert for adding a word to the end of a line. When tested on a new task, users tended to choose the method learned with text that had similar content to the text in the current task, even though either method would work.

2. *Mapping Between Conceptual Structures*

Once an example is retrieved as a candidate for analogy, a mapping must be identified between its conceptual structure and the structure of the current task. If the current task involves something that *is-a program* and *has-name Cricket Graph,* then a mapping can be made to an example involving something that *is-a program* and *has-name Word.*

In our implemented models, declarative knowledge such as *has-name Word* is expressed in attribute-value notation. "Has-name" is an attribute of programs, and one of the programs our model knows about has its "has-name" slot filled with the value "Word." The knowledge structure representing a program or any other real-world object can involve many attribute-value relationships. The facts may be episodic (*was started*) instead of categorical (*is-a program*), but the structures still need to be similar if analogy is to succeed.

3. *Using the Mapping to Describe a New Action*

The mapping, once identified, is applied to the example action that accomplished the example task. In our case, *has-name Word* in the example task was mapped to *has-name Cricket Graph* in the current task. Applying this mapping to the example action, *double-click* X, where X *is-a icon* and *has-name Word,* must be transformed into *double-click Y,* where Y *is-a icon* and *has-name Cricket-Graph.*

The success of this step requires that the previous step identified the appropriate attributes and values. In our example, the model must identify the name of the program as a meaningful attribute in the current and the example tasks, and it must identify the name of the icon as a meaningful attribute in the example action. This seems trivial, but there is no *a priori* reason why these text strings should be the linking feature. The relationship might be one of location, like handles on spigots, or of meaningful iconic representation, like the labels on a car dashboard.

Additionally in this step, the mapping may need to incorporate chains of facts between the example task and the example action, to make features of the action meaningful. Imagine that the current task is to make cool air come out of a car's vents, and the driver knows that the red button of the climate control turns on the heater. To propose pushing the blue button for air conditioning requires the additional knowledge that red stands for hot and blue stands for cold. Identifying this linking knowledge requires search through all knowledge associated with the example.

4. *Testing the New Action*

Having identified an action that may accomplish the current task, the model takes that action and observes the result. Since analogy is an inductive process, there is no guarantee that the action will have the desired effect. For our model's task, the action of double-clicking on Cricket Graph produces a clear indication of success: the program's start-up greeting. In other cases success is not so certain. The user who has the goal of printing a document may choose "print" from a menu, only to be shown a dialog box of print options. This is at best a step on the way to success, if the user can recognize it as such and continue.

5. *Learning New Rules*

The model learns two kinds of rules. First, in the process of analogical reasoning, it may learn rules that will make it easier to perform similar reasoning when another new program has to be started. The exact form of these general rules depends on the implementation. Second, the model learns the very specific rule already described: "If you want to start Cricket Graph, then double-click on the Cricket Graph icon." (Note that this rule cannot be learned reliably until the action's effect is confirmed.) The model does not learn a general rule for starting an arbitrary new program.

5.4.5 Version 4: Details

Again in this model all knowledge has been represented using just four generic Types: object, action, effect, and proposition. The bulk of the knowledge is represented as propositions about objects, where each proposition has a subject, a relation, and an object. For example, the Cricket Graph icon is an object with internal identifier 05, and one of the propositions describing it is: (p09 is-a proposition, subject 05, relation has-name, object Cricket-Graph).

In addition to declarative knowledge represented in Working Memory structures, the model represents certain basic procedural knowledge as productions. Productions fire when their condition side matches the current goal. For most productions in the model, the result of firing was to add new information to Working Memory. Many productions also alter the goal or the stack of goals, and a few productions have the special function of causing physical movements that interact with the outside world. Productions are divided into those that are specific to the interface and those that are not. For those that are, our interest in producing a learnable system demanded that we tell a story of how they were acquired and that we implement that acquisition process if possible.

The behavior of the model closely tracks the abstract model presented in the previous section. Here again, only the first action (double-click the data icon in the Finder) is described, although the implemented model can work all the way through to click on the "OK" button.

The model begins the task by looking for matches between objects on the screen and elements of the instructions. Instructions are represented in subject-relation-object form, such as "subject me, relation start, object Cricket Graph." The first such overlap to occur is between the name of the Cricket Graph icon and the object slot of the instruction. Having identified an overlap, the model focuses on the Cricket Graph icon as a candidate for action.

Once an object becomes the focus of attention, the model must propose an action to be taken on that object. It does this with an interface-specific production, which is a procedural encoding of the object's affordance and its effect. If this production already exists, it simply fires and the model continues. If the production does not exist, the model attempts to form an analogy between the current situation and experience with a similar object in the past. In the case of the Cricket Graph icon, the model recalls a similar experience with the Microsoft Word icon; it decides to form an analogy to that experience, and it restructures its working memory and sets its goal to force an impasse. This situation -- an appropriate memory structure and an impasse -- is exactly what is required for ACT-R's built-in analogy mechanism to engage. The analogy mechanism then creates a new production, available now and in the future, to propose actions on application icons in the Finder.

In ACT-R, analogy is the only way that new productions can be formed, and analogy will take place only when no other productions apply. The rather convoluted way of inducing analogy in our model reflects the inability of the ACT-R implementation to treat analogy as a member of the conflict set to be considered along with applicable productions.

The action proposing production has two effects: (1) it proposes a specific action: double-click on the icon; and (2) it places in Working Memory an envisionment of the effect of that action: the Cricket Graph program has been started. The envisioned effect may, as in this case, reflect interface-specific knowledge: double-clicking starts a program. But in some cases, no such knowledge is available. What, for example, is the result of selecting a menu item that has never been selected before? Menu items may be nouns, verbs, or modifiers, and there is no reliable prediction of what an item will do. In such cases, the system performs what we term "wishful envisionment." It knows from the instructions what effect is desired, and it places that effect into Working Memory as the envisioned result.

Having envisioned the effect of the proposed action, the model now checks that effect for two conditions. First, is it consistent with the instructions? Of course, it will be if it is a wishful envisionment, but it may not be if it is derived from interface-specific knowledge. Second, is the envisioned effect something that still needs to be achieved? To confirm this, the model consults a task-status record that will be updated when actions are performed. Conceptually the two conditions can be checked in parallel, although the implementation does them serially, in random order.

If both conditions are satisfied, the model decides to achieve the effect by performing the action. It does this by setting the effect as its goal and then placing into Working Memory an imagined past interaction in which that goal was satisfied by taking the proposed action. Because there is no preexisting production that will fire to satisfy the goal, the ACT-R analogy mechanism again engages. It forms an analogy between the goal and the imagined past interaction, identifies the mapping, uses that mapping to define the new action, and produces new production rule. The rule has the current goal as its condition, and it fires immediately, causing the model to take the proposed action.

After the action is taken, the model must check to see that it caused the desired result. Any decision based on wishful envisionment has a chance of being wrong, and even decisions based on known interface conventions will fail for programs that do not adhere to the standards. Therefore, after the action is taken, the model compares the perceived change in the world to the envisioned result. If the world matches the envisionment, the model updates the task status and continues. If the match fails, the model leaves the task status unchanged. In this case, it must also disable the production that has just been formed, to prevent the production from firing in future, similar situations. It does this by altering an element in Working Memory that we call the "fuse" for the newly formed production. The fuse is a declarative proposition referenced in the production's condition, and altering the fuse will permanently prevent the production from firing.

5.5 The Soar Model

The Soar model was developed and programmed by Richard Young (Rieman et al, 1994). Soar is a cognitive architecture developed by Newell and his colleagues and proposed as a "universal theory of cognition." Soar is essentially a production-system architecture, which includes a declarative working memory but no long-term declarative memory. Behavior in Soar is structured around "problem spaces." When the current goal cannot be satisfied by productions that fire in the current problem space, the Soar system impasses and selects another problem space to resolve the difficulty. Soar includes a universal method for learning, called "chunking," that records the results of problem solving and makes them available when similar situations arise in the future.

The Soar model of the Cricket Graph task performs essentially the same high-level reasoning sequence as described for the final version of the ACT-R model, but with some important lower-level differences. Note that the Soar model is implemented only to handle the task of starting the Cricket Graph program by double-clicking the icon.

The Soar model begins with the goal of starting the Cricket Graph. Because no known production can satisfy this goal, the system impasses. To resolve the impasse, it chooses the analogy problem space. This is a problem space specifically created for this model; unlike ACT-R, Soar does not include analogy as a default feature.

In the analogy problem space, Soar takes the strategy of imagining how to solve a similar problem with another software application. Specifically, it imagines how it would solve the problem of starting Microsoft Word. This is something it knows how to do. It then uses the declarative trace of this activity as the basis for "deliberate analogy." Quite simply, it maps the process for starting Word onto the current problem by substituting "Cricket Graph" for every instance of "Microsoft Word." This yields the suggestion that it should double-click the Word icon, which it does.

The model learns certain things about the process, using Soar's built-in chunking mechanism. First, it learns a specific production that describes how to start Cricket Graph: double-click the Cricket Graph icon. Second, it learns a support production that might be useful if it had to figure out how to start yet another application; this support production embodies the result of imagining the start-up of Microsoft Word, so the step of imagining it need not be performed again. The Soar model does not learn a general production that tells it how to start any application by double-clicking that application's icon.

5.6 Soar vs. ACT-R: Problem Solving vs. Situated Cognition

In comparing the Soar and ACT-R models, the most significant difference is the handling of goals. The Soar model begins with the goal of starting Cricket Graph, whereas the ACT-R model begins with the generic goal of figure-out-what-to-do. The issue here is not whether or not Soar could handle the entire problem, from presentation of instructions to final problem solving. Soar models have been produced that interpret instructions and transform them into goals, and such a instruction-following model can be assumed as part of a more complete system. The issue, instead, is that the Soar and ACT-R models are designed to deal with the world in fundamentally different ways. The Soar model must work with a single, well-defined goal, while the ACT-R model always draws information from both the interface and its list of remembered instructions in order to decide what to do next. In terms of the discussion at the beginning of this chapter, the Soar model is a problem solver, while the ACT-R model engages in situated cognition.

Note that this difference between the models is not necessarily inherent in the difference between Soar and ACT-R. In fact, our ACT-R model takes an unusual approach as compared to models supplied as examples with the ACT-R implementation. It may be that Soar could be programmed to perform situated cognition isomorphically to the ACT-R model; it appears even more likely that the reverse is true: ACT-R could take the same general approach as Soar.

Notwithstanding their fundamental differences, the ACT-R and Soar models share some interesting details. Both models rely on imagining (or "envisioning") the effect of an action, although they use this information somewhat differently. The ACT-R model imagines the proposed action, then treats that as past experience; the Soar model imagines the action needed for another program, then analogizes to that.

The models also share the common feature of maintaining most of the interface-specific knowledge as production rules, not as declarative facts. Thus, for example, the ACT-R model does not include a general declarative fact that icons can be double-clicked, only a production that proposes double-clicking them. This approach is not surprising for Soar, which has no long-term declarative memory, but it is unexpected for ACT-R.

The models differ significantly in the kinds of general rules they learn. In ACT-R, where analogy is part of the basic system, the model learns interface-specific generalities, such as the rule for proposing double-click as an action associated with icons. In Soar, where analogy is simply another approach to problem solving, the model learns rules that will assist similar analogical reasoning. Additionally, the Soar model's analogy process can handle only simple substitution, "Cricket Graph" for "Word," whereas ACT-R's analogy mechanism can include chains of knowledge, as in the red/blue temperature example.

Neither model learns a general rule of the form, "To start any Mac program, double-click its icon." We were able to build models that learned such a rule in both architectures, but those models failed to satisfy our concerns about learnability. The models could learn the general rule, but we couldn't justify the knowledge they needed as a prerequisite.

Finally, it should be noted that neither of the implemented models treats the entire process of analogical learning. The importance of surface similarity is only weakly represented, and there is no facility for restructuring knowledge. Only the ACT-R model checks the action's effect before learning a new rule.

5.7 Discussion

The novice construction-integration model along with the early ACT-R models demonstrate that label following is a powerful but limited strategy. It can pull the naive user through a narrowly constrained path (assuming the user's task description closely matches the labels provided by the interface), but it fails when multiple choices must be compared, or when the user must consider the current action as it relates to past interactions.

Quite simply, label following operates only when all necessary information is contained on the screen in the immediate present. For more sophisticated interactions, both the past and the future must be considered. Past actions must be remembered so they can be avoided (don't double-click the Cricket Graph icon twice), supplemented (click OK now that the relevant parts of the dialog box have been handled), and checked (now that the dialog box has disappeared, are the results as expected?).

Remembering past actions is relatively simple, as shown in the earlier ACT-R models. The key to sophisticated interactions, however, seems to be envisioning future states. The final ACT-R model and the Soar model both rely on this facility.

An additional lesson is that handling fallback actions is surprisingly difficult in the ACT-R framework. Fallbacks here are defined to include actions such as clicking "OK" or pulling down a different menu when no menu matches an instruction. In general, fallbacks must be taken whenever the instructions don't provide a clear label-following path, or when that path has already been taken and the next step is not specified by the interface. Handling fallbacks requires checking the entire interface to be sure nothing matches an instruction. Indeed, it may even require search to some depth, scanning all pulldown menus, for example. Situations such as these clearly call out for an associative component to the model, one in which fallback actions gradually become more attractive as other actions fail, instead of simply counting off all possible actions and then immediately choosing the fallback.

Once the need for a fallback is evident, the appropriate action itself must be selected, with options that may range from clicking "OK" to looking through more menus to asking someone for help. These are exactly the kinds of activities observed in the diary studies, activities which more experienced users had clearly prioritized as to predicted effectiveness. A still more sophisticated model of exploratory learning would include productions that implemented this range of fallback activities.

Chapter 6

LABORATORY OBSERVATIONS OF EXPLORATION

To provide a deeper understanding of how users handled the difficulties revealed by the theoretical analysis and modeling efforts, users were observed in a controlled laboratory situation as they attempted a task that required them to perform task- and problem-oriented exploration in a display-based interface.

The field work described in Chapters 3 and 4 indicated that users typically explore new computer applications in the context of a defined task. The theoretical analysis described in Chapter 2 and the cognitive modelling efforts described in Chapter 5 predict that certain forms of behavior will characterize those explorations. To test the predictions of the theory and the model, we observed the task-oriented exploratory behavior of users in a laboratory situation. Subjects were asked to perform the same Cricket Graph task that had been investigated in the modelling work described in Chapter 6.

The experiment described in this chapter was designed primarily by the author. Marita Franzke assisted in setting up and monitoring the experimental procedure, and subjects were run by Marita Franzke and Troy Davig. The larger purpose of the experiment was to investigate training and transfer effects between two versions of the Cricket Graph program. Preliminary results of that investigation are reported in (Franzke & Rieman, 1993), and a detailed analysis of more extensive work on this topic can be found in (Franzke, in preparation). The Method section of this chapter is excerpted from (Franzke & Rieman, 1993); the results described in this chapter are reported here for the first time.

6.1 Method

The experiment investigated users' interactions with versions 1 and 3 of the Cricket Graph software. Subjects were initially given a task to solve with Cricket Graph 1, then tested on a similar task using Cricket Graph 1 or Cricket Graph 3. Only the initial training sessions with Cricket Graph 1 are reported here.

6.1.1 Subjects

Twenty-two undergraduates from the University of Colorado subject pool participated in this experiment for class credit. Subjects had an average of 2.4 years of experience with Macintoshes but no experience with Cricket Graph. Subjects were randomly assigned to several conditions for the training and transfer experiment. For the purposes of the current analysis, subjects are treated as single group, since all were initially given a task with Cricket Graph version 1. One subject was removed from the data set because of equipment failure and one because of inappropriate experimenter intervention early in the experiment, leaving an n of 20.

6.1.2 Tasks and Procedure

6.1.2.1 Cricket graph Tasks

The experimental task given to subjects consisted of two main phases: (a) creating a new graph from a data file provided to the subjects, and (b) editing the default graph to match it to a given sample graph. The data was provided in a Cricket Graph data file that the subjects could use without further modifications. Subjects also received a sample graph that differed from the default created by Cricket Graph in a number of dimensions. To create a near-perfect duplication of the sample format, subjects would have to change the style of the graph title and axes labels, the Y-axis label, the data-point symbol and plot-line style, the style of the legend label, the location of the legend, the style of tick marks, and the range of the axes.

Subjects had to decide on the types and the order of modifications they wanted to attempt. A perfectly duplicated format was impossible, since the sample graph had been created with a different program than Cricket Graph. The freedom of subjects to choose their own goals, combined with the imperfect sample, gave us the opportunity to observe goal instantiation and management in a relatively natural task environment, without enforcing a particular grain size or order of subgoals.

The instructions were written in a small HyperCard stack, which included general instructions, a sample graph, and details about the data, graph type, and data-axis mappings. Subjects could page through the three instruction cards by clicking on labeled buttons, and the stack was designed to ensure that the subject looked at each card at least once before beginning the task. An opened folder in the Macintosh finder contained the appropriated Cricket Graph version and the data file. The folder window was partially covered by the stack and could be activated and brought to the front by clicking on it. The HyperCard stack was accessible during the whole task. This setup enabled us to record precisely when subjects consulted the sample graph or the written instructions for further directions.

6.1.2.2 Warm-Up Tasks

Before subjects were presented with the main graphing tasks, they had to be warmed up to two procedures: thinking aloud and using Macintosh Multifinder. Initial screenings of the tasks warned us that our subject population might not be accustomed to moving back and forth between different application windows. To give them some practice in toggling between the HyperCard instructions and the task window, subjects first had to perform a brief word-processing task using Microsoft Word, in which they were asked to format document to match a sample text given on HyperCard. This task and the instructions had a similar format to the Cricket Graph task and familiarized subjects with the Multifinder environment.

While subjects were doing these tasks they were encouraged to think aloud (Lewis, 1982). With this method we were able to collect additional information on what goals subjects were pursuing throughout the task, as well as which interface object they were attending to. Prior to any tasks involving the computer, subjects were warmed up to thinking aloud with a brief verbal problem-solving task. All computer tasks were performed on a Macintosh IIcx computer with a 13-inch color monitor.

Procedure. Upon arrival subjects were instructed in the thinking aloud procedure and did the verbal problem-solving task. After this the experimenter directed their attention to the Macintosh display and asked them to read through the instructions aloud and follow the instructions on their own. Subjects were informed that the experimenter would remind them to verbalize their thoughts but would not answer any questions or provide any hints about the task. Subjects were given ten minutes to complete the word-processing task.

After the word-processing task was completed, the experimenter brought up the instructions for the graphing task as well as the folder containing the data and the Cricket Graph application. Subjects were not allowed to use the Cricket Graph manual. The lack of a manual reflected the finding in diary studies and interviews (Chapters 3 and 4) that novice users were often unable to find what they needed in documentation, even if it was available. Similarly, telling the subjects that the experimenter would not answer questions forced the user to perform some exploration, simulating a natural situation in which asking questions requires more initial effort than searching for a command in the interface. If subjects persistently failed to discover critical interaction techniques and continued to explore the same parts of the interface, the experimenter would volunteer brief hints, such as "try working directly on the graph" (intended to divert attention from the menus).

Subjects were allowed ten minutes to complete the first Cricket Graph task. If they had been unable to accomplish any of the editing subtasks, because they did not discover the methods that would lead to their completion, they were given a brief demonstration by the experimenter. Essentially, the experimenter showed the subject how to accomplish the immediate goal(s) that the subject had been working on. This manipulation ensured that all subjects entered the transfer task with some knowledge about successful interface techniques in one Cricket Graph version.

After the experimenter's demonstration, subjects were transferred to the other version of the software, where they performed a second task. The complete experimental procedure lasted approximately 45 minutes and was videotaped by a camera that recorded the interactions visible on the monitor, along with the thinking-aloud protocols.

6.2 Label Following

The ACT-R and construction-integration models of cognitive behavior, described in Chapter 5, both predict that much of the exploratory behavior in the Cricket Graph task will reflect a label-following strategy. In label following, the user attempts to identify visible controls with labels that match the current goal. When such a control is located, the action it affords is taken.

A point within the Cricket Graph task where label following is unequivocally predicted by the models is the situation where Cricket Graph has been started, the data file has been loaded, and the next appropriate step is to create the default graph. The goal should be to "create a graph," and a menubar item labelled "Graph" is the correct control to select. The behavior of each subject was examined at this point in the interaction.

6.2.1 Results

Of the 20 subjects, 3 subjects moved the mouse cursor directly to the graph menu and pulled it down, without pointing at or clicking on any other part of the interface. Another 4 subjects moved the cursor to the graph menu without clicking anything else, but they did pass over other parts of the interface, as if using the mouse cursor to focus their visual search. The behavior of these subjects falls entirely within the predictions of the model, which implicitly includes the action of scanning the visible screen.

Another 5 subjects actually pulled down other menubar menus or clicked on tools within the toolbar, extending their search a level deeper than that allowed by simple visual scan of the current screen. The model does not explicitly predict these actions. Interestingly, 3 of the 5 subjects pulled down the Graph menu at least once, then checked other menus, before returning to select from Graph.

The 8 remaining subjects initially engaged in actions that showed evidence of strategies or goals not predicted by the model. One of these subjects explicitly stated the goal of "seeing what kind of stuff" the interface offered; he spent about 15 seconds looking at all the menus before returning to Graph. Five subjects restarted Cricket Graph one or more times after the data spreadsheet was visible, perhaps thinking that some kind of graphics window should appear. One of these subjects explained that he was "trying to get the graph part," while another stated that she was "looking for those things up there," pointing to the menubar area; she may have been looking for a Microsoft-style toolbar. All of these subjects eventually found the Graph menu. Finally, 2 subjects attempted to manually draw the graph into the spreadsheet window; both of these subjects found the Graph menu after the experimenter suggested they look in the menubar.

6.2.2 Discussion

To summarize, 7 out of 20 subjects (35 percent) selected the Graph menu through label-following behavior exactly as predicted by the model. An additional 5 subjects (25 percent) used a label-following strategy supported by an active search to one level of controls beyond those displayed on the initial screen. The remaining 8 subjects (40 percent) showed the ability to use label following, but that behavior was initially blocked by misconceptions or alternate goals.

The behavior of the subjects who supplemented their label-following strategy with active search is especially interesting. This strategy may be a learned response to applications that have menubar titles with low or overlapping semantic content, such as the "Font" and "Format" menus found in many word processors. In such a situation, the items on the pulldown menus carry most of the semantic information needed to drive successful label following, and an unaided visual search of just the top-level items on the menubar will seldom yield success. This strategy has important implications for design and evaluation of interfaces. In the cognitive walkthrough evaluation method (Polson, Lewis, Rieman, Wharton, 1992), we advise the evaluator to examine the screen for an easily identified correct control label. The evaluator is also told to check for "foils," incorrect labels that might be a good match to the current goal. The behavior observed in this experiment suggests that evaluators and designers should extend their attention beyond the visible screen to include those parts of the interface that are easily accessible by common actions, such as scanning the pulldown menus.

Also of interest is the behavior of the five subjects who restarted Cricket Graph after the spreadsheet data window appeared. Within the framework of the cognitive walkthrough, this can be simplistically described as a failure to provide adequate feedback; the users were evidently unsure that they had actually started the graphics program, even though Cricket Graph provides a start-up banner message that is visible for more than a second. But the behavior also has strong implications for label following. Here is a situation in which the correct label was clearly visible, yet a significant number of users were initially unwilling to look for it, because their expectations as to the result of the previous action had not been met. After one or more restarts of Cricket Graph, each of these users evidently felt satisfied that the program had started, at which point they shifted to a label following strategy and successfully located the correct menu item.

Finally, note that two of the subjects were completely stalled because they had formed the inappropriate subgoal of manually creating a graph. For these users as well, the clearly visible label "Graph" had no impact. Indeed, these users were in a worse situation than the Cricket Graph restarters, since the interface offered no help in redefining their goal. Both users required a hint from the experimenter before they could continue.

6.3 Menu Search Strategies

As soon as the default graph is created, subjects must begin to modify it to match the sample graph shown in the instructions. The modifications required include changing the text of one axis label and the graph title, changing the type style of several items, and changing the line style and the style of the data points. Each of these changes can only be accomplished by double-clicking on the screen object to be modified. Experience with pilot experiments showed that few subjects would discover this method; almost all would first explore the menus looking for a Font or LineStyle option. Eventually subjects would discover the graphic Toolbox, but typically they tried these options after a fair amount of menu search. Because this search was doomed to fail, this point in the interaction was ideal for observing extended menu search strategies.

The formal analysis predicted that a guided depth-first search with iterative deepening (gDFID) would be the most effective way of locating the controls needed within the interface. With the Macintosh interface, this would be realized by first scanning the menubar, then pulling down each of the menus, then selecting attractive menu items to observe their dialog boxes or effects, then delving progressively further into potentially useful dialog boxes. The protocols of the 20 subjects were examined for evidence of gDFID or alternate strategies.

6.3.1 Results

6.3.1.1 gDFID

The search behavior observed did show evidence of gDFID search, but not in the simple, immediate manner predicted. Only one of the subjects obviously scanned each item in the menubar before first pulling down the menus. And once the menus had been pulled down and examined, the next level of search was almost never pursued before scanning the menus again. The next level of search after scanning should have been to try attractive menu items. There are 43 pulldown menu items in the Cricket Graph program that could be selected, and it was clear that subjects used semantic behavior at this level to limit the search space, as predicted by the "guided" component of gDFID. Seventeen of the subjects tried 7 or fewer pulldown menu items; the largest number of items selected by any subject was 16. However, this limitation of the search was maintained in spite of the fact that the menu items selected did not yield success with the subjects' goals. Subjects failed to extend their search when the semantically guided search failed.

6.3.1.2 Deepening Attention

How, then, did the subjects spend their exploratory time? For most of them, the time was filled with another form of deepening search. That search was of memory, activated by iterative scans over the same menus with increasingly greater attention to each item, and especially to pulldown menus whose options did not have effects that were obviously inappropriate to the current goal. The iterative deepening of attention was clearly observed in 15 of the 20 protocols, as evidenced by sliding the mouse cursor down a menu that had only been visually scanned on the first pass (11 subjects), by increasing time spent looking at a pulldown menu (10 subjects), and by reading the menu items out loud on a later pass (3 subjects).

6.3.1.3 Spatial Scan Strategy

The gDFID algorithm suggests scan strategy of checking attractive menus first, then falling back to look into menus that seem less likely to contain the necessary options. A competing scan strategy for the Macintosh menu structure is spatial: left-to-right across the menubar, top-to-bottom for each pulldown menu, and left-to-right, top-to-bottom (reading-scan style) for dialog boxes. This strategy requires the least physical work and also reduces the cognitive load of remembering which menus have been checked. Subjects showed a combination of these strategies. Eight of the 20 protocols showed full left-to-right or right-to-left scans at one point in the protocol, with the scans typically starting in the middle of the menubar. Another 5 showed partial scans, often beginning in the middle of the menubar with the Graph menu and scanning to the right, skipping the File and Edit menus, as well as the grayed-out Data menu. Even the subjects who scanned the full menubar at first would typically reduce the breadth of their scan in later iterations of the search, sometimes to the point of concentrating on one or two menus, usually including Goodies.

There was also evidence of top-to-bottom scans of pulldown menus, but this was much weaker. A few subjects slid the cursor all the way down some of the menus or read the menu items out loud from top to bottom. No subject did this with all of the menus.

6.3.1.4 Label Avoidance

The menu scan strategy and the deepening attention behavior both showed evidence of a strategy that could be termed "label avoidance." In this behavior, interface controls that are clearly inappropriate to the current task are avoided, while all others are considered. This is an elimination strategy as opposed to the selection strategy of "label following." Label avoidance was hard to distinguish in the protocols from simple random scan strategies. However, it was reflected in much larger amount of time spent by most subjects in the Goodies menu as opposed to the File menu. With specific menu items, the verbal protocols gave some evidence of the strategy. One subject selected Interpolate from the Curve Fit menu, saying: "I don't know that so I'll try that." Later he paused over the Switch Axes item under Goodies, then went on, saying: "No, I don't want to switch axes."

A possible result of the label avoidance strategy was the popularity of the Add Depth menu item under Goodies. Add depth makes the graph's frame appear to be three dimensional, if the frame is turned on. Since the frame is turned off by default, the Add Depth option usually had no effect. The option was selected by 9 of the 20 subjects, often with a verbal comment to the effect of, "Let's see what this does." A tenth subject paused the mouse cursor over the option twice but avoided the selection at the last moment: "No, I don't want to add depth." The option was selected while subjects were pursuing a variety of goals, although the goal of making the line bolder seemed more likely to attract it.

6.3.2 Discussion of Menu Search Strategies: a Dual Search Space

The most important guiding strategy for the majority of the subjects observed was the deepening attention strategy. This was supplemented to a greater or lesser degree by label avoidance (i.e., avoiding the file menu) and limited trials of options that could neither be selected nor rejected on the basis of their labels.

This deepening of attention can best be understood in terms of the construction-integration model of expert behavior described by Kitajima and Polson (1993). Their model predicts that even experts will make errors with an interface if they do not allow themselves to retrieve all the information needed about an interface object. The protocols reported here appear to show novice users making a similar judgement. When the first, cursory examination of a set of interface objects does not activate sufficient information to suggest an action, the users examine the objects again with longer and more focused attention, forcing deeper activation of information in long-term memory and increasing the likelihood that the facts needed to make a decision will be retrieved. For some users the iterative scan strategy is made even more sophisticated by reducing the attended set of objects to those that could not be clearly eliminated by the earlier scan, a "label avoidance" technique that is discussed in greater detail below.

The menu search strategy, then, actually involves two coordinated searches, one of the interface and one of memory. Each component of this dual search affords the potential for iterative deepening. The interface search can be deepened by trying lower levels of controls, while the memory search is deepened by allowing longer times for activation.

6.4 Goal Setting Strategies

The instructions given to the subjects did not provide detailed, step-by-step instructions concerning the changes that had to be made to transform the default graph into the required format. The subject's task was to discover differences between the default graph and the sample, post goals to reduce those differences, and decide when those goals were satisfied. There was no guarantee that all differences could be eliminated, and indeed, some features of the sample graph, such as the background pattern, could not be matched in Cricket Graph. The protocols, therefore, provide an opportunity to observe how users set goals, the tenacity with which they maintain those goals, and the abilities they have to supplement their higher-level goals with subgoals or temporary alternative goals.

6.4.1 Results

6.4.1.1 Weakly Posted Goals

The formal analysis predicted that weakly posted goals would be an effective strategy for exploration. Such goals would be stated and pursued, but if success was not achieved with a reasonable amount of time and effort, they would be dropped. Of the 20 protocols, 13 showed evidence of goals being dropped after efforts to achieve them failed.

There was a wide range of attention times per goal. At one end of the spectrum, one subject posted and then dropped several goals after working on each for less than two minutes: "It won't let me do this so I'm not going to worry about it," he said. Another subject maintained a single goal for 8 minutes with little evidence of progress, then continued to work on the same problem for another 3 minutes after the solution path became obvious.

Although difficult goals appeared to be disabled, it was common for subjects to discover a solution path to a previously dropped goal and to then reactivate that goal. It was also common for subjects to achieve success at a new goal and then restate the goal that had been dropped earlier.

6.4.1.2 Opportunistic Sidegoals

A strategy closely related to weakly posted goals is the use of opportunistic goals, or "sidegoals." Opportunistic sidegoals were recognized and acted on by 8 of the subjects. These sidegoals included goals that had not been clearly stated earlier in the protocol as well as goals that had earlier been only partially satisfied. They were "opportunistic" in the sense that they were activated when some interaction with the interface made their solution obvious. In each case, the goal preceding the sidegoal was returned to after the sidegoal was achieved. A common example of this in the protocols was moving the legend. A subject would accidentally drag the legend a few pixels while trying to click on screen objects to change fonts or line sizes. The subject would then drag the legend into its correct position and return to the earlier goal of setting font or line size.

6.4.1.3 Error-Correction Subgoaling

The third form of goal management observed in the protocols was the correction of errors. In attempting to change the line size and plot-point style, subjects would often try the curve-fit options, which placed the text of an equation in the graph window, or they would work with the graphics toolbox, leaving messy circles and arrows on the plot. Thirteen of the 20 subjects set the subgoal to correct these errors before continuing with their current goals. As with the higher-level goals, these would be weakly posted by some subjects and dropped if they were not satisfied in a short time.

6.4.2 Discussion of Goal Setting Strategies

There seems to be no rational way to decide how long a goal should be pursued before it is dropped, and the widely varying behaviors of the subjects may reflect this lack of a good solution. It was interesting to note, however, the ease with which most subjects "pushed" their current goal onto a goal stack and then retrieved it after correcting an error, taking care of an opportunistic sidegoal, or looking back at the instructions. Similarly, goals that had been dropped as unsatisfiable seemed easy to retrieve when new information about the interface suggested ways to achieve them. Within the short period spanned by our protocols, typically 10 to 15 minutes, subjects showed very few problems with forgetting goals once the goals had been clearly stated.

6.5 Discussion

In analyzing the protocols it was surprisingly difficult to associate specific patterns of behavior with success or failure at the task, perhaps because the laboratory situation lacked realistic resources for supplementing exploration with help from other sources, such as manuals and more knowledgeable users. A more detailed description of the behaviors of two users, who represent composit pictures of two behavioral extremes observed in the subjects, will help bring this point home.

User A carefully checks the instructions, finds several changes that need to be made to the default graph, and explicitly posts the goal of changing the line and plot symbol styles. He begins his investigation with a slow scan of each of the pulled-down menus, an unusual technique that evidently assumes that he will not recognize the appropriate menu item with just a quick glance in a new application. He then goes back through the menus and carefully tries several options that could reasonably be expected to advance his goal: Add Format, a different style graph from the Graph menu, Simple from Curve Fit, which he follows by Remove from Curve Fit to clean up the screen. Eventually he gives up on the menus and tries to work with the toolbox. He has no success there either, and he soon gives up, saying he's tried everything he can think of. The experimenter then tells him how to double-click on screen objects. With this added piece of information he quickly completes the task, posting and achieving a series of subgoals that transform the graph.

UserB, in brief, shows very little ability to rationally search the interface. He pulls down random menus, selecting items that don't even seem to have any arguable overlap with the task, items such as "Show Clipboard," and "Add Depth." He jumps back to the instructions without fully investigating the interface, then gets lost in the interface as he tries to find the default graph again. He adds Curve-Fit information and doesn't seem to notice that the screen has changed. But his erratic behavior eventually pays off -- he double-clicks a screen object, possibly in a clumsy attempt to drag or single-click it, and discovers the key technique that allows him to edit the default graph. Having discovered the technique, however, he lacks the task-management skills to cleanly identify necessary subgoals and accomplish them. He fixes the font size on one axis, neglects to take care of it on another, and fails to notice that the legend is misplaced. Overall, his performance is poor -- but he did it without help.

Which approach would be most productive in a real-world situation? The diary studies and interviews suggest an answer. Without the experimenter present, User A would likely have given up after a reasonable amount of time and gone to the manual or another user for help, allowing him to finish his work in a controlled fashion. User B's erratic approach yielded such an incomplete search that he would have been unable to make a reasonable decision about when to give up, had he not serendipitously discovered the double-click trick.

Chapter 7

SUMMARY AND FINAL DISCUSSION

The research revealed little evidence of task-free exploration in an everyday work context, but found task-oriented exploration, supported by manuals and personal help, to be widespread. An analysis of the theoretical difficulties of exploration and the cognitive strategies used to support it helps explain these results and suggests ways in which interfaces themselves could be made more explorable.

As described in Chapter 1, there was reason to believe when this research began that exploration might be a productive and enjoyable strategy for learning about computer applications. However, there was also reason to believe that it had not been successfully applied by many users. The goal of the research was to provide a better understanding of exploratory behavior, where that behavior was defined to include a range of activities, from task-free investigations to task-oriented exploratory learning. It was hoped that this understanding would suggest interface design strategies in support of exploration.

7.1 Review of Research Strategy

The research reported in this dissertation approached the issue of exploratory learning on several fronts. Formal models suggested absolute constraints and practical difficulties associated with exploratory behavior. Field investigations identified areas in which users were able to overcome those constraints in the real world. Cognitive models suggested low-level behaviors that could effectively navigate novel interfaces. And laboratory protocols validated and enriched the models.

The sequence of the chapters roughly indicates the information flow from one research effort to the next, but to some extent all efforts overlapped and fed their results into each other. This partially parallel approach was perhaps not as clean as the idealized research paradigm attributed to the students of animal behavior in the first chapter, but it yielded a rich body of converging evidence describing interactive computer users' actual behavior and the theoretical reasons for that behavior. The next section summarizes that data.

7.2 Summary of Results

The theoretical investigations in Chapter 2 revealed that exhaustive task-free exploration of an interface appeared to be a difficult, almost insurmountable problem. The exploration space of even a simple program was simply too large to approach without a guiding task. However, the analysis concluded that task-oriented exploration was a more tractable problem, and suggested that users might approach task-free exploration by setting weakly defined, opportunistic goals, associated with "microtasks" defined in response to the offerings of the interface. A potentially effective algorithm for task-oriented exploration, guided depth-first search with iterative deepening (gDFID) was proposed.

In Chapters 3 and 4, field studies and associated interviews were described that attempted to gather high-level evidence of exploratory behavior and strategies. This focus was part of a wider net that was cast to capture all forms of ongoing learning associated with computer systems. The most striking result of these studies, from the point of view of this research effort, was the paucity of task-free exploration observed. In both their behavior and their interview statements, the majority of users seemed to have recognized the difficulties of task-free exploration and to have made the decision to avoid this time-consuming and often unproductive behavior.

On the other hand, these same users were almost universally willing to engage in at least some amount of task-oriented exploratory learning when a real-world task took them beyond the bounds of their current computing skills. They typically supplemented their on-line explorations with searches through printed manuals and interactive requests for help from other people.

Because task-oriented exploratory learning was found to be a prevalent behavior, the cognitive modeling efforts reported in Chapter 5 focused on that activity. A series of models of increasing complexity were developed that suggested ways in which the user could handle the problem solving and learning involved with accomplishing a new task through exploration. These simple models addressed only the leading edge of the user's behavior, i.e., the investigations within the interface itself, without looking at how manuals or other resources might be used to aid the investigation. The models demonstrated that simple label following, which is a behavior with very low cognitive costs in terms of memory or problem solving, could lead the user through a new section of the interface, if the interface structure and labels closely matched the task.

Chapter 6 described low-level exploratory behavior of users in a laboratory situation. The users' protocols were examined for evidence of the strategies predicted by the formal analysis and the cognitive modelling, particularly weakly posted and opportunistic subgoals, gDFID search, and label following. All of these strategies were found, but not in the simple form described by the earlier analysis. Label following was often supplemented with an active investigation of hidden control options, and gDFID was combined with an iteratively deepening search of memory.

7.3 Discussion

7.3.1 Learning Strategies and Support

Task-free exploration was not found to be a preferred strategy for learning an interface. It is difficult and time consuming, and users are unwilling to allocate time for activities that are not clearly work related. Even experienced users, whose greater knowledge of the interface and richer long-term memory structures could potentially support task-free exploration, generally find this behavior unproductive. Task-oriented exploratory learning fares better. In the context of current or impending problems, users are willing to spend time exploring the interface for solutions, especially when that exploration is combined with the use of manuals and access to personalized help.

Keeping in mind the informants' stated concern with time and productivity, we can identify two important functions of manuals and other resources in support of exploratory learning.

First, these resources provide an *overview* of the program and its mapping to the user's task. This overview includes the invaluable evidence that the current task actually can be performed with the given program. Without this assurance, any exploration has the potential of becoming an exhaustive exploration of the system's capabilities, needed to ensure that the sought-for feature does not exist. The overview also describes the overall task structure. The user who knows that Cricket Graph automatically creates graphs showing the relationships of values in a spreadsheet is in a far better position to explore the interface than is the user who thinks Cricket Graph is a manually operated drawing program. This kind of information not only aids in label following, it also helps in envisioning the effects of controls and checking for task completion.

Second, the support of a good manual or a more experienced user provides *details* that the user can use as fallbacks when exploration fails. This will often happen when the user's terminology for an operation does not match the terminology used in the interface. As Furnas, Landauer, Gomez, and Dumais (1987) have observed, this "vocabulary problem" is inevitable in interface design. Whatever term the designer chooses for an interface control, there will be many users who expect some other term. For display based interfaces the issue is one of recognition and not generation, but protocols of users working with the Cricket Graph task show that this is still a problem. A well organized and indexed manual or a few words from an experienced user may overcome these difficulties, constraining a large exploration space to a manageable size.

7.3.2 Suggestions for Improving Interfaces and Computing Environments

It was hoped that this research effort would suggest design guidelines for more easily explorable interfaces. Reviewing the research results suggests that there are two approaches to this goal.

7.3.2.1 Modifications to the Interface

On the one hand, interfaces themselves can be modified. Given users' preference for task-oriented activities, there seems to be little benefit in supporting task-free behavior by structural modifications to the software such as those suggested in the formal model of Chapter 2. However, task-oriented exploration might be supported by methods that overcome the difficulties identified in the cognitive modelling and laboratory work. Two important modifications recognize the fact that users are comfortable with scanning pulldown menus but reluctant to actually select an item that doesn't clearly match to their current goal.

The first modification is to use labels that clearly predict the results of selecting the control, using terminology that a task and user analysis has shown to be appropriate to the user population. The second modification is to place as much of the system's main functionality as possible on controls that can be accessed by shallow active scanning behavior, that is, by looking at the interface and sliding the mouse cursor over menus and buttons. One way to accomplish this would be to temporarily display the effect of a menu item or button when the user pauses the mouse cursor over the item. For example, holding the mouse cursor over "Open..." in the "File" menu for more than 200 ms could cause the dialog box for opening files to appear. The dialog box would disappear when the user moved the mouse to another menu item. (The 200 ms delay reduces computational load and avoids annoying the user who knows what menu item he or she is moving to.)

7.3.2.2 Modifications to the Extended Interface

The stronger message of the research is to look beyond the interface in designing explorable systems. In the short view, this means considering what we have termed "the extended interface": manuals, on-line help, help lines, and even marketing documents, all of which help the user to acquire a overview of the tasks supported by the interface, as well as the general solution paths associated with those tasks (Lewis & Rieman, 1993).

A specific area of interest is on-line help. Although opinions vary with the user and the computer system, the great majority of informants reported in this study, as well as many users the author has spoken with informally, find on-line help difficult or impossible to use. One reason seems to be that the on-line system fails to provide the appropriate mix of overview and detail, where overview describes the general mapping between the user's task and the interface, while details provide specific information, such as the name of the menu item to perform a subtask the user knows must be performed. A second possible reason, related to user's reluctance to take exploratory actions that require explicit undo actions, is that using on-line help forces the user to deal with one more set of system interactions and states, on top of the task-oriented system state that is already a problem. The perceptual and cognitive overload may simply compound the problem.

From a broader point of view, the research suggests that the work environment in which a system is embedded has a powerful influence on the user's success with the software. A productive environment is likely to include systems personnel who respond rapidly to help requests, as well as a cooperative work attitude where workers are encouraged to spend a few minutes sharing their skills with others when those skills are needed.

7.3.3 Caveats

It is appropriate here to identify significant limitations of the research and conclusions just summarized. The research investigated the behavior of a relatively well-educated, computer sophisticated sample of users, most of whom have had long-term access to modern computing facilities including display-based personal computers and electronic mail. The behavior of these users, in particular the reliance on manuals and help from systems support and other users, strongly reflects both the information richness of their environment and the confidence and sophistication of the individuals. Nardi (1993) and MacKay (1990) found similar collaborative work in studies of spreadsheet and CAD users. The results of the current research, then, describe a situation that may be widespread in the culture of today's workers in large organizations, but the result may not apply to individuals who work at home or in very small offices.

Furthermore, this is an area of changing individual and social behavior. In offices where the author worked with users of word processors and dedicated typesetting equipment in the mid-1980s, the dominant learning approach was to rely on formal training. The widespread availability of personal computers, and the widespread need for local expertise, were able to create today's collaborative computer work culture in roughly a decade. Changing conditions in the working environment could produce a very different approach in the future. For example, the informants' current reliance on manuals indicates that they have had success with manuals in the past, while their avoidance of on-line help indicates an experience of failure. As popular programs evolve, particularly word processors and electronic mail, these experiences may change, and users may come to view other resources or behaviors as most productive in completing their work-related tasks.

In short, the research reported here applies theoretical considerations to a narrow time-slice of real-world behavior. Extending the theoretical considerations to users in other situations is a meaningful enterprise. Extending the practical conclusions requires careful attention to the new situation and the user population involved.

7.4 Future Work

This findings reported here suggest ideas for additional research in several areas. There is a need to focus more sharply on individual learning events, in order to develop a finer-grained understanding of how users distribute their information-seeking activities among the interface, on-line help, manuals, personal interactions, and other resources. This might help designers create better programs and manuals, but it could also suggest improvements to interpersonal workplace arrangements that would better support "distributed cognition," in the sense that the full understanding of the system is spread across the skill sets of many users.

The findings concerning problems with on-line help suggest a specific area on which further investigation needs to focus. What is the cognitive cost of using on-line help in the context of a particular system problem? What are the perceptual difficulties associated with intermixing the system display and the help display? The hypertext capabilities of on-line help, as well as the ease with which it can be distributed and updated, make it potentially a very attractive medium for providing information to the user, if the problems can be identified and overcome.

Finally, the modelling work only touches the surface of an experienced user's skills and strategies for investigating a new interface. As computer skills become a standard part of many worker's job qualifications, it becomes increasingly important to understand how experienced users extend their knowledge to new systems and system upgrades. Further modelling and empirical work can inform this question.

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. (1993) *Rules of the Mind.* Hillsdale, NJ: Lawrence Erlbaum.

Archer, J. and Birke, L., Eds. (1983). *Exploration in Animals and Humans.* Berkshire, England: Van Nostrand Reinhold.

Aho, A.V., Hopcroft, J.E., and Ullman, J.D. (1983). *Data Structures and Algorithms.* Reading, MA: Addison-Wesley.

Bellotti, V.M.E. (1990). A framework for assessing applicability of HCI techniques. *Proceedings of Interact90, 3rd IFIP Conference on Human- Computer Interaction,* Cambridge, England, August, 1990.

Berlyne, D.E., (1960). *Conflict, Arousal and Curiosity.* New York: McGraw-Hill.

Bias, R. (1988). User interface walkthroughs with representative users and usability experts. Paper read at the symposium "Human Factors Methods (th)at Work," Annual Meeting of the Human Factors Society (Anaheim, CA, October, 1988).

Bovair, S., Kieras, K. E., and Polson, P. G. (1990). The acquisition and performance of text editing skill: A production system analysis. *Human Computer Interaction,* 5, 1-48.

Brain, P. F. (1988). Ethology and experimental psychology: From confrontation to partnership. In Blanchard, R.J., Brain, P.F., Blanchard, D.C., and Parmigiani, S. (Eds.), *Ethoexperimental Approaches to the Study of Behavior.* Dordrecht/Boston/London: Kluwer, pp. 18-27.

Broadbent, D. (1982). Task combination and selective intake of information. *Acta Psychologica, 50,* 253-290.

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence, 47,* 139-159.

Card, S.K., Moran, T.P., and Newell, A. (1983). *The psychology of human-computer interaction.* Hillsdale, NJ: Erlbaum.

Carroll, J. M. (1982). The adventure of getting to know a computer. *IEEE Computer* 15 (11), 49-58.

Carroll, J.M. (1990). *The Nurnberg Funnel.* Cambridge, MA: MIT Press.

Carroll, J.M., Mack, R.L., Lewis, C.H., Grischkowsky, N.L., and Robertson, S.P. (1985). Exploring a word processor. *Human Computer Interaction, 1,* 283-307.

Carroll, J.M., Mack, R.L., and Kellogg, W.A. (1988). Interface metaphors and user interface design. In M. Helander (Ed.), *Handbook of Human-Computer Interaction.* Amsterdam: Elsevier (North-Holland). pp. 67-81.

Carrol, J.M., and Mazur, S.A. (1986). Lisa Learning. *IEEE Computer 19* (10), 35-49.

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: MIT Press/Bradford Books, pp. 80-111.

Charney, D., Reder, L, and Kusbit, G. (1990). Goal setting and procedure selection in acquiring computer skills: a comparison of tutorials, problem solving, and learner exploration. *Cognition and Instruction, 7,* 323-342.

Chi, M.T.H., Feltovich, P.J., and Glaser, R. (1981). Categorization and representation of physics problems by experts and novices. *Cognitive Science, 5,* 121-152.

Christ, R.E. (1975). Review and analysis of color coding research for visual displays. *Human Factors, 17,* 542-570.

Day, H.I., Ed. (1981). *Advances in Intrinsic Motivation and Aesthetics.* New York: Plenum.

Doane, S., Kintsch, W., and Polson, P.G. (submitted). UNIX command production: What users must know. *Human Computer Interaction.* Also available as ICS Technical Report #90-1, Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0345.

Draper, S.W. (1985). The nature of expertise in UNIX. In B. Shackle (ed.), *Proceedings of First IFIP Conference on Human-Computer Interaction — Interact'84.* Amsterdam: North-Holland, 465-472.

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.

Ericsson, K.A., and Simon, H.A. (1984). Verbal reports as data. *Psychological Review, 87,* 215-251.

Fantino, E., and Abarca, N. (1985). Choice, optimal foraging, and the delay-reduction hypothesis. *Behavioral and Brain Sciences 8* , 315-330.

Fischer, G. (1987). Cognitive view of reuse and redesign. *IEEE Software, Special Issue on Reusability, 4* (July 1987), 60-72.

Floyd, C., Wolf-Michael, M, Reisin, F.-M., Schmidt, G., and Wolf, G. (1989). Out of Scandinavia: Alternative approaches to software design and system development. *Human-Computer Interaction 4* , 253-350.

Fowler, H. (1965). *Curiosity and Exploratory Behavior.* New York: Macmillan.

Franzke, M. (in preparatino). Exploration, acquisition, and retention of skill with display-based systems. Ph.D. Dissertation, University of Colorado.

Franzke, M. and Rieman, J. (1993). Natural training wheels: Learning and transfer between two versions of a computer application. *Proceedings of Vienna Conference on Human Computer Interaction 93.* Sept. 20–22, 1993. Vienna, Austria.

Furnas, G.W. , Landauer, T.K., Gomez, L.M. , andDumais, S.T. The vocabulary problem in human-system communication. *Communications of the ACM, 30* (Nov. 1987), 964-971.

Gibson, E.J. (1988). Exploratory behavior in the development of perceiving, acting, and the acquiring of knowledge. *Annual Review of Psychology, 39,* 1-41.

Gick, M., and Holyoak, K. (1983). Schema induction and analogical transfer. *Cognitive Psychology, 15,* 1-38.

Görlitz, D. (1987). Exploration in an everyday context: situational components and processes in children and adults. In Görlitz, D., and Wohlwill, J.F. (Eds.), *Curiosity, Imagination, and Play.* Hillsdale, NJ: Lawrence Erlbaum Associates. pp. 106-125.

Gould, J.D., Boies, S., and Lewis, C. (1991). Making usable, useful, productivity-enhancing computer applications. *Communications of the ACM 34* , 74-89.

Hayes-Roth, B., and Hayes-Roth, F. (1979). A cognitive model of planning. *Cognitive Science, 3,* 275-310.

Holland, J., Holyoak, K., Nisbett, R., and Thagard, P. (1986). *Induction: Processes of Inference, Learning, and Discovery.* Cambridge, MA.: MIT Press.

Holyoak, K.J., and Koh, K. (1987). Surface and structural similarity in analogical transfer. *Memory and Cognition, 15,* 332-340.

Houston, A., Kacelnik, A., and McNamara, J. (1982). Some learning rules for acquiring information. In *Functional Ontogony,* David McFarland, Ed., Pitman, London, pp. 140-190.

Howes, A. (1994). A Model of the Acquisition of Menu Knowledge by Exploration. To appear in *Proceedings of CHI'94 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery.

Johnston, W.A., and Dark, V.J. (1986). Selective attention. *Annual Review of Psychology, 37,* 43-75.

Kamil, A.C., and Balda, R.P. (1985). Cache recovery and spatial memory in Clark's Nutcrackers. *J. Exp. Psych., Animal Behavior Processes 11* , 95-111.

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: 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. (1988). The role of knowledge in discourse comprehension: A construction-integration model. *Psychological Review*, 95, 163-182.

Kintsch, W., and Ericsson, A. (1991). Memory in comprehension and problem solving: A long-term working memory. ICS Technical Report 91-13, Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0430.

Kitajima, M., and Polson, P.G. (1992). A computational model of skilled use of a graphical user interface. *Proceedings of CHI'92 Conference on Human Factors in Computing Systems.* New York: ACM, pp. 241-249.

Klahr, D. and Dunbar, K. (1988). Dual space search during scientific reasoning. *Cognitive Science, 12,* 1-48.

Koran, J.J. Jr., Morrison, L., Lehman, J.R., Koran, M.L., and Gandara, L. (1984). Attention and curiosity in museums. *Journal of Research in Science Teaching, 21,* 357-363.

Korf, R.E. (1985). Depth-first iterative deepening: An optimal admissible tree search. *Artificial Intelligence, 27,* 97-109.

Korf, R.E. (1988). Search: A survey of recent results. In Shrobe, H.E., and the Association for Artificial Intelligence (Eds.), *Exploring Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann. pp 197-237.

Laird, J., Newell, A., and Rosenbloom, P. (1987). SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33, 1-64.

Landauer, T.K. (1988). Research methods in human computer interaction. In *Handbook of Human-Computer Interaction,* M. Helander, Ed., Elsevier Science, Amsterdam, 1988, pp. 905-928.

Lave, J. *Cognition in Practice.* Cambridge/New York: Cambridge University Press.

Lewis, C. (1982). Using thinking aloud protocols to study the "cognitive interface." Research Report RC 9265, IBM Thomas J. Watson Research Center, Yorktown Heights, NY.

Lewis, C. (1988). Why and how to learn why: analysis-based generalizations of procedures. *Cognitive Science,* 12, 211-256.

Lewis, C., and Polson, P.G. (1991). Exploration and learning in interactive systems. Proposal submitted to the National Science Foundation.

Lewis, C.H., Polson, P.G., Wharton, C., and Rieman, J. (1990). Testing a walkthrough methodology for theory-based design of walk-up-and-use interfaces. *Proceedings of CHI'90 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery, 235-241.

Lewis, C. and Rieman, J. (1993). *Task-Centered User Interface Design: A Practical Introduction.* Boulder, Colorado: Shareware electronic publication, available via anonymous ftp to ftp.cs.colorado.edu.

Mackay, W. (1990). Users and customizable software. A co-adaptive phenomenon. Ph.D. dissertation. Sloan School of Management. MIT, Cambridge, MA.

Malone, T. W. (1982). Heuristics for Designing Enjoyable User Interfaces: Lessons from Computer Games. *Proceedings of the Conference on Human Factors in Computing Systems* (Gaithersburg, MD). New York: Association for Computing Machinery, 63-68.

Mannes, S.M. and Kintsch, W. (1991). Routine computing tasks: Planning as Understanding. *Cognitive Science ,15,* pp.305.

Miller, D. B. (1985). Methodological issues in the ecological study of learning. In *Issues in the Ecological Study of Learning,* T.D. Johnston and A.T. Pietrewicz, Eds., Lawrence Erlbaum, Hillsdale, N.J., pp. 73-95.

Molich, R. and Nielsen, J. (1990) Improving a human-computer dialogue: What designers know about traditional interface design. *Communications of the ACM,* 33, 338-348.

Muncher, E. (1989). The acquisition of spreadsheet skills. Unpublished master's thesis, University of Colorado, Department of Psychology, Boulder, CO.

Nardi, B. (1993). *A Small Matter of Programming.* Cambridge, Mass: MIT Press.

Neal, L.R. (1987) Cognition-sensitive design and user modeling for syntax-directed editors. *Proceedings of CHI+GI '87 Conference on Human Factors in Computing Systems and Graphics Interfaces,.* New York: Association for Computing Machinery, 99-102.

Neisser, U. (1967). *Cognition and Reality.* San Francisco: W.H. Freeman

Newell, A. (1990). *Unified Theories of Cognition.* The William James Lectures. Cambridge, MA: Harard University Press.

Newell, A., and Simon, H. (1972). *Human Problem Solving.* Englewood Cliffs, NJ: Prentice Hall.

Newell, A. Shaw, J.C., and Simon, H.A. (1958). Elements of a theory of human problem solving. *Psychological Review, 65,* 151-166.

Nielsen, J., Mack, R.L., Bergendorff, K.H., and Grischkowsky, N.L. (1986). Integrated software usage in the professional work environment: evidence from questionnaires and interviews. *Proceedings of CHI'86 Conference on Human Factors in Computing Systems,.* New York: Association for Computing Machinery, 162-167.

Nielsen, J. and Molich, R. (1990) Heuristic evaluation of user interfaces. *Proceedings of CHI'90 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery, 249-256.

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: Basic Books.

Payne, S.J., Squibb, H.R., and Howes, A. (1990). The nature of device models: The yoked state space hypothesis and some experiments with text editors. *Human-Computer Interaction, 5,* 415-444.

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., Atwood, M.E., Jeffries, R., and Turner, A. (1981). The processes involved in designing software. In J>R> Anderson (Ed.), *Cognitive skills and their acquisition.* Hillsdale, NJ: Erlbaum.

Polson, P.G. and Lewis, C.H. (1990). Theory-based design for easily learned interfaces. *Human-Computer Interaction, 6,* 191-220.

Polson, P.G., Lewis, C., Rieman, J., and Wharton, C. (1992). Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces. *International Journal of Man-Machine Studies.*

Renner, M.J. (1991) Neglected aspects of exploratory and investigatory behavior. *Psychobiology, 18,* 16-22.

Rieman, J. (1993). The diary study: A workplace-oriented tool to guide laboratory studies. *Proceedings of InterCHI'93 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery. pp.321–326.

Rieman, J., Lewis, C., Young, R.H., and Polson, P.G. (in press, 1994). "Why Is a Raven Like a Writing Desk?" Lessons in Interface Consistency and Analogical Reasoning from Two Cognitive Architectures. *Proceedings of InterCHI'94 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery.

Rieman, J., Davies, S., Hair, D.C., Esemplare, M, Polson, P.G., Lewis, C. (1991) An automated walkthrough (demonstration). *Proceedings of CHI'91 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery. Expanded version available as ICS Technical Report # 90-18, Institute of Cognitive Science, University of Colorado, Boulder, CO 80309-0345.

Rolleston, J. (1970). *Rilke in Transition: an Exploration of his Earliest Poetry.* New Haven: Yale University Press.

Rosenbloom, P.S., Laird, J.E., Newell, A. (1991) A preliminary analysis of the SOAR architecture as a basis for general intelligence. *Artificial Intelligence, 47,* 289-326.

Ross, B.H. (1984). Remindings and their effects in learning a cognitive skill. *Cognitive Psychology, 16,* 371-416.

Rosson, M.B. (1984). Effects of experience on learning, using, and evaluating a text-editor. *Human Factors, 26,* 463-475.

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

Russell, P.A. (1983) Psychological Studies of exploration in animals: a reappraisal. In Archer, J. and Birke, L. (Eds.), *Exploration in Animals and Humans.* Berkshire, England: Van Nostrand Reinhold, pp. 22-54.

Shneiderman, B. (1983). Direct manipulation: a step beyond programming languages. *IEEE Computer,* 16 (8), 57-69.

Shrager, J. (1985). Instructionless learning: Discovery of the mental model of a complex device. Unpublished Ph.D. dissertation, Carnegie-Mellon University.

Shrager, J., and Klahr, D. (1986). Instructionless learning about a complex device: The paradigm and observations. *International Journal of Man-Machine Studies, 25,* 153-189.

Simon, H.A. (1973). The structure of ill-structured problems. *Artificial Intelligence, 4,* 181-201.

Simon, H.A. (1990). Invariants of human behavior. *Annual Review of Psychology, 41,* 1-19.

Simon, H.A. and Greeno, J.G. (1988) Problem solving and reasoning. In Atkinson, R.C., Herrnstein, R.J., Lindzey, G., and Luce, R.D. (Eds.), *Stevens' Handbook of Experimental Psychology (2nd ed.),* Vol. 2, pp. 589-672.

Singley, M.K., and Anderson, J.R. (1988). A keystroke analysis of learning and transfer in text editing. *Human Computer Interaction, 3,* 223-274.

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

Sperling, G., and Dasher, B.A. (1986). Strategy and optimization in human information processing. In Boff, K.R., Kaufman, L., and Thomas, J.P. *Handbook of Perception and Human Performance,* Vol. I, pp. 2-1 - 2-65.

Spohrer, J., Soloway, E., and Pope, E. (1985). A goal/plan analysis of buggy Pascal progams. *Human Computer Interaction, 1,* 1985, 163-207.

Suchman, L. A. (1987) *Plans and Situated Actions.* Cambridge, England: Cambridge University Press.

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

Sweller, J, Chandler, P., Tierney, P., and Cooper, M. (1990) Cognitive load as a factor in the structuring of technical materials. *Journal of Experimental Psychology: General, 119,* 176-192.

Tomback, D.F. (1978). Foraging strategies of Clark's Nutcracker. *Living Bird 16* , 123-161.

Treisman, A. (1986). Properties, parts, and objects. In Boff, K.R., Kaufman, L., and Thomas, J.P. *Handbook of Perception and Human Performance,* Vol. II, pp. 35-1 - 35-70

Vander Wall, S.B. (1982). An experimental analysis of cache recovery in Clark's Nutcracker. *Animal Behavior 30* , 84-94.

Vander Wall, S.B., and Balda, R.P. (1977). Coadaptations of the Clark's Nutcracker and the piñion pine for efficient seed harvest and dispersal. *Ecological Monographs 47* , 89-111.

Voss, H-G. (1987). Differences between exploration and play in terms of action theory. In *Curiosity, Imagination, and Play.* Görlitz, D., and Wohlwill, J.F. (Eds.). Hillsdale, NJ: Lawrence Erlbaum. pp.152-177.

Winston, P.H. (1984). *Artificial Intelligence* (2nd ed.). Reading, MA: Addison-Wesley.

Wixon, D., Holtzblatt, K., and Knox, S. (1990). Contextual design: An emergent view of system design. In *Proceedings of the Conference on Human Factors in Computing Systems* . New York,: Association for Computing Machinery, 331-336.

Wohlwill, D. (1987). Curiosity, imagination, and play: communality and interrelationships. In Görlitz, D., and Wohlwill, J.F. (Eds.), *Curiosity, Imagination, and Play.* Hillsdale, NJ: Lawrence Erlbaum Associates. pp. 2-21.

Young, R.M., Green, T.R.G., and Simon, T. (1989). Programmable user models for predictive evaluation of interface designs. *Proceedings of CHI'89 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery, 15-19.

Young, R.M., and Whittington, J. (1990). Using a knowledge analysis to predict conceptual errors in text-editor usage. *Proceedings of CHI'90 Conference on Human Factors in Computer Systems.* New York: Association for Computing Machinery, 91-97.

Appendix. Summaries of Eureak Reports

Key

R: Read the paper manual, U: Used on-line help, T: Tried different things until it worked, St: Stumbled onto by accident, A: Asked (phone or in person), Se: Sent e-mail or posted to newsgroup, N: Noticed someone else doing it, O: Other

Informant 2.

2.1/A. Restructured directories in Unix -- asked systems support for help.

2.2/A. Saved article from News group -- asked student (me) for help.

2.3/0. Subscribed to new News group. O=Remembered instructions from previous day, tried it.

Non-computer Eurekas:

2.4/AT. Figured out how to spell word, asked someone, then tried it out (recorded as O) by writing word down, saying it out loud.

2.5/0. Found phone number, O=looked in old mail, then notes.

Informant 3.

3.1/A. Posted article on bulletin board from e-mail, called systems for help.

3.2/T. Needed to log OUT of payroll system, tried things until it worked -- could have called or looked at manual, but didn't because of time constraint: "When am I supposed to read all this stuff? [referring to all the manuals for office systems]"

3.3/TA. Payroll program wouldn't print to printer, tried things, then called software support; they said call hardware, S did.

3.4/T. Needed borders on Table cells, read mfgr + 3rd-party manuals, tried things out. Didn't use index or contents -- all Table stuff in one place.

3.5/T. Wanted to paste one thing into several places; tried pasting more than once after a single copy. Knew clipboard concept, just "occurred to S" that it might work.

Non-computer Eurekas

3.6/T. Needed to add new account on copy machine. Systems people not available immediately, so tried things -- never could get it right.

Eurekas outside of log period:

3.7/TN. Tried, noticed: Wanted to move several files at one time into a folder. Remembered seeing it done, remembered hearing someone say "shift-click" while they did it. Couldn't remember what to do then, so experimented (what to click, when to release keys, what to drag).

Informant 4.

4.1/O. Looked for databases on Carl for a thesis. About 25-min through terminal, added time in library. Was looking for non-book entry, eventually looked under book and found it.

4.2/TUN. Worked with Unix "last" and "head" and "grep." Got basics out of Man pages, then tried it. Had seen colleague do something similar.

Eurekas outside log period:

4.3/RT. Wanted to match upper case only in Grep. Checked Man, didn't find answer; guessed that quotes would do it and tried: it worked.

Informant 5.

5.1/RT. Got a form entered for FoxBase, Read and Tried. Has 4 manuals, knows similar database (dbase). So knows what can be done, but not how to find it or what it's called. Does some browsing of menus to find things.

5.2/UTSeO. Downloaded OzTex and printed manual. Four hours of work: U, T, Se (e-mail to colleague), O (knew ftp). Had to learn binhex, stuffit. Had to ftp Unix->Unix, then Unix->Mac. Didn't ask except one specific question about bibtex/latex.

5.3/TSt. Couldn't get ftp & binhex stuff to work on some files, so tried Xferit on a Mac (hunting for any possibility). Ftp stuff is hard, because it's Unix style. Mac stuff is easy to learn. Didn't have manuals.

5.4/RT. Got FoxBase to do complex input checking -- read paper manual, tried things. Knew another database that would do this easily, but this was difficult.

5.5/TO. Bib program wouldn't read more than seven files, but error message didn't explain that was the problem. Tried things and O=worked with another user (who didn't initially know the answer either -- cooperative problem solving episode, so not scored as "A").

5.6?RT. Got a simple, "hello-world" program to run. Had to figure out several subparts. Read manual and tried things.

5.7/St. Found that "&" would do macro-expansion in a Fox command. Reading manual for something else, and stumbled onto this. Had hoped it would be possible (so was prepared for the accident). I'll score this as a "St".

5.8/RO. Figured out how to select multiple items on the Mac. S scored as "O": knew it should be possible (based on other op sys knowledge), along with "R" the paper manual.

5.9/RTO. Figured out how to make an NCSA-TCP set file and get it to come up on the desktop. Read manual -- didn't help. Tried things. Knew it had been done on another machine, so no doubt it could be done.

5.10/R. Got OzTex to include mac pict files from MacDraw clipboard. Read paper manual.

5.11/RT. Found that & macro doesn't work w/ variable names longer than 10 chars. Read manual, but it didn't give this info. Experimented (tried things) to debug.

5.12/RT. Learned that MacDraw won't put arrow on curved line. Read manual, tried things, finally concluded failure. Expected feature to work since it works in another program.

5.13/T. Tried to transfer files from Mac to Unix using XFer. Gave up after 15 minutes (trying things).

5.14/TASe. Problem with MacBibTex -- it can handle only three files. Tried different things, asked colleague in office, sent e-mail to program developers and got updated version that fixed the problem.

5.15/RT. Wanted to use OzTex on a LaTex file instead of a Tex file. Read paper manual and tried things. Config file had to have options in a certain order -- trial and error was only way to discover it.

Eurekas outside of log period

5.16/AN. Learned that "//t" on library on-line database would limit search to titles. Asked librarian how to find a periodical, watched while she did it

5.17/RO. Wanted to set Penman (natural language program in Lisp, with Mac interface) to use a specific option: set-system-mode-manual. Had seen the option in the interface, but never used it. (Sometimes browses systems.) Read manual, but that was a waste. Used on-line "find-file/code" to look into the source code, since the feature was undocumented.

5.18/T. Another Penman problem. Penman developers evidently had a similar problem, had already solved it.

Informant 6.

Non-computer Eurekas

6.1/T. Programmed VCR to record upcoming program. Couldn't remember order in which to push buttons, but figured it out by trying things.

6.2/A. Needed authorization code to send FAX. Asked supervisor, who gave it to S.

6.3/O. Showed someone else how to make transparancies on copier.

6.4/TA. Tried to put chain back on wheel of bicycle. Didn't succeed. Tried different things, then asked someone to help. Still no luck.

Informant 7

7.1/StA. HCIBib (on-line database) came up in conversation, S' asked for info.

Eurekas outside of log period

7.2/A. Tried to find volume info for usenet newsgroups. Asked collegue who was down the hall (knew student who would be more likely to have answer, but he wasn't down the hall).

7.3/O. Got and explored new on-line text corpus. Just looked at readme files and in various directories. (Not trying to do anything with it yet, so coded as "O" not "T".)

Informant 8

8.1/R. Wanted to see form of function. Used pprint, looked up form of argument in the paper manual. (Note that "paper manual" for this informant is always *Franz Lisp, the Reference.*)

8.2/R. Same as 8.1, for different function.

8.3/R. Same as 8.1, for different function.

8.4/R. Same as 8.1, for different function.

8.5/R. Had name conflict problem, read manual to help debug problems.

8.6/R. No Lisp routine called "before." Could have looked through code to find definition (i.e., it is defined in own code), but easier to check the paper manual.

Informant 9

9.1/TR. Needed to fit large spreadsheet onto single page. Did it by reducing fonts and changing orientation. First tried different things, but couldn't find reorient function. So looked in manual from a class he had taken.

9.2/RT. Learned that endnote does the underlining of titles automatically. Read manual and tried things. (Printed out all the material he had been entering for several days, saw how it printed.)

Non-computer Eurekas

9.3/T. Found out how to stack originals for duplex copies on the copier.

9.4/R. Learned that correct bibliographic style is to underline conference proceedings.

Informant 10

10.1/TStA. Had problem reading directory in emacs that had links. Tried things, gave up, later ran into someone and asked them, then when problem was carefully demonstrated it disappeared. Weird emacs-Unix interaction.

10.2/RUTA. Tried to load unix-based authoring system three times. Several versions on system, Unix confused about which to use. Had to go to manual for Lisp. Called system support, got that to run. Then ran into another problem while working through the next steps in a README file.

Eurekas outside computer.

10.3/O. Used copier. Tried things, it failed. Someone notice problem and helped. (coded as "O")

10.4/RTA. Programmed the phone to dial common numbers. Tried things, but it didn't work. Asked someone, also read paper manual. Part of the problem was that functionality was literally hidden under a card that contained pencil annotations of the programmed numbers.

Informant 11

11.1/RT. Wrote a batch file to pop-up an electronic form of the diary log on screen every half hour. Looked up relevant statements in table-of-contents of batch editor technical reference manual.

11.2/UT. Scanned picture wouldn't open in editing program. Checked on-line help, found nothing. Saved picture in different format and tried again; it worked.

11.3/RUT. Wanted to crop photo. Looked for "crop" tool or masks in help and manual, found nothing. Tried selecting portion and copying to another file; didn't work because resolution deteriorated. Finally used "line" tool to draw white line around edges.

11.4/RUT. Used lots of new features of Corel draw: extension, blends, perspective, layers. Tried lots of buttons. Also recalled watching video (yesterday) that came with the program. This is an update of a program Informant has used before.

11.5/T. Wanted to work on a paper which I had on tape but not on disk; but had also removed the software used to make the tape. Restored old backup software from new backup tape, then used old software to retrieve paper from old tape.

11.6/RT. Produced on-line version of Eureka slip using scanner. Figured out how to insert date/time automatically and add check symbol. Basically a matter of getting formatting to within aesthetic threshold.

11.7/RUT. Paste together small scanned pieces into a bigger image. Looked in manual and help to see if there's any way to increase the dimensions of an image or to join images side-by-side; couldn't find anything. Eventually figured out I'd have to open a new blank image of the correct size and paste all the pieces into it and position each one.

11.8/RUTA. [Eureka occurred over two days] Day 3: Tried to link text from Word file into new presentation software. Couldn't actually be done. Best way was to paste data into grpahics file and link that to presentation software. (NB: I didn't actually have any immediate reason to want to do this, just thought I might in future. Day 5: Called MS tech support. Evidently there was something wrong with the "registration database." Reinstalled part of Word. Now I can embed Word objects in other applications, ubt in CorelShow they still display only as an icon and not as text. Will call tech support again tomorrow.

Eurekas outside of log period

11.13./U. Needed to insert a date so it won't be updated, in Word on PC. Poked around in help topics relating to fields until I found it. Since this is actually what I normally want when I insert a date, fixed date macro so it always does this. (You have to insert a "date field" and then "unlinmk" it.)

11.15/T. Tried to use a photo of beach pebbles for Windows background. Scanned photo; loaded it into Photo-Pain; converted grayscale scanm to color; tried out various new tools in Photo-Paint to alter the colors and even the image; saved it as a bitmap and loaded it as "wallpaper." Colors came out different. Gave up.

11.16/RT. Excel wouldn't open files from Quattro. Used manual to find a file format both programs would read: Lotus. Restored quatro from tape (it had been deleted), translated files into Lotus format, deleted Quattro again, loaded Lotus-format files into Excel.

Non-computer Eurekas

11.9/O. Coral manual too thick. Added improvised index tabs.

11.10/T,A (also outside log period). With colleague, attempted and failed to help speaker set up overhead projector. Someone else walked by, they asked her and she showed them how to do it.

11.11/T. Too much documentation kicking around desk. Made a plastic pouch from a sheet-protector and taped it to side of computer for quick-reference guides and keyboard templates.

11.12/T. (also outside log period). Nut coming loose on lawnmower handle. Nutdriver too small. Used a crescent wrench and pliers.

11.14/RT. (also outside log period). Wanted to see if it were possible to make an ice cream soda at home. Looked for a recipe (in cook boods) but couldn't find one. However found a recipe for chocolate syrup to be used in milshakes. Adapted it for microwave. Made some of that and combined it with tonic water, milk & a scoop of ice cream. Came out not very authentic but pretty good.

Informant 12

12.1/TO. Colleague volunteered information on making Mac window to VAX display more than 24 lines. Informant later discovered that you have to set the option every time you log in, or put the information in login.com.

12.2/T. Learned how to make large cells around small font text in MS Word table.

12.3/N. Found out how to edit a button icon in HyperCard by noticing someone else do it.

12.4/T. Learned there is no "transpose" function in Cricket Graph III. Had to paste every cell by hand. (Tried things until sure that it *didn't* work.)

12.5/R. Learned how to save a screen shot and resize it in MacDraw. Read "The Macintosh Bible."

12.6/TA. Tried to eject Mac disk, couldn't because "item is still in use" message appeared. Overrode by shutting off computer. Later talked to colleague and learned that I need to Quit MS Word when I've worked on a document on the disk.

Informant 13 -- There was no informant 13.

Informant 14

14.1/N. Lotus can do "joins" on a field. (Not that I'll ever use it.) Colleague was doing it with Lotus. I did it for him in R:Base.

14.2/A. Discovered only had 12 copies of Lotus on network. It doesn't let you in when 12 people are on.

14.3/T. Setting the "rules" off in "Set-up" menu doesn't apply for importing -- must set rules off in "import/export" menu. Tried translating to older version, other experiments.

Informant 15

Non-computer Eurekas

15.1/St. Found out about metal-filled extracellular recording electrodes -- "hold" cells for long periods of time, easy to make. Was talking about related topic with colleague, she mentioned metal-filled electrodes as a viable option for records.

Informant 16

16.1/T. Wanted to print calender using PC. Looked at all available software: scheduler, spreadsheet, Windows, paint, etc. Was in a hurry so didn't look at manual. Accustomed to Mac, so if menus don't work, gives up fast -- "no time for manuals (or DOS!)."

16.2/A. Learned how to access dedicated customer information system on minicomputer workstation.

16.3/A. Learned how to run credit-card payment. Batch mode.

16.4/A. Learned how to use dedicated billing/payment system. Included learning to process customer payments by cash, checks, credit cards. Need to login to system, then use lots of codes. Need additional knowledge [overview of what's happening].

16.5/A. Learned to access classified advertising system and perform some operations.

16.6/TA. Attempted an upload, PC->Mainframe, but it failed. (Couldn't find the problem.)

16.7/A. Learned to use advertising info system to run schedules, schedule ads, run balance.

Non-computer Eurekas

16.8/A. Needed to use fax, didn't know how. Found out that fax tries 5 times, then gives up -- with no indication that it's failed.

16.9/A. Learned to work the switchboard.