The Psychology of Human-Computer Interaction

Stuart K. Card
Thomas P. Moran
Xerox Palo Alto Research Center
Allen Newell
Carnegie-Mellon University

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business

Preface

Designing interactive computer systems to be efficient and easy to use is important so that people in our society may realize the potential benefits of computer-based tools. Our purpose in this book is to help lay a scientific foundation for an applied psychology concerned with the human users of interactive computer systems. Although modern cognitive psychology contains a wealth of knowledge of human behavior, it is not a simple matter to bring this knowledge to bear on the practical problems of design—to build an applied psychology that includes theory, data, and methodology.

This book is our attempt to span the gap between science and application. We have tackled a small piece of the general problem. With respect to computer science, we have focused on the task domain of text-editing and similar types of highly interactive systems. With respect to psychology, we have focused on the notion of the expert user's cognitive skill in interacting with the system, especially the temporal aspects of the interaction. We have constructed an empirically-based cognitive theory of skilled human-computer interaction in this domain. This theory is our keystone for linking science and application. On one side, we have shown that the theory is a consistent extension of the science of human information-processing. On the other side, we have simplified the theory into practical engineering models, which are the tools for designers to apply the theory. Thus, in addition to putting forth specific psychological models in this book, we have tried to make clear the general framework of an applied psychology, in which these models are but prototypical examples.

THE AUDIENCE FOR THIS BOOK

Interest in the topic of human-computer interaction is shared by people from a range of disciplines. We believe this book makes contact with the specific interests of all of these disciplines. For instance:

1. An Applied Information-Processing Psychology

A scientific psychology should not only help us to understand our own human nature, it should help us in our practical affairs. In educating our children, it should help us to design environments for learning. In building airplanes, it should help us to design for safety and efficiency. In staffing for complex jobs, it should help us to discover both the special skills required and those who might have them. And on and on. Given the breadth of environments we design for ourselves, there is no limit to the number of domains where we might expect a scientific knowledge of human nature to be of use.

The domain of concern to us, and the subject of this book, is how humans interact with computers. A scientific psychology should help us in arranging this interface so it is easy, efficient, error-free—even enjoyable.

Recent advances in cognitive psychology and related sciences lead us to the conclusion that knowledge of human cognitive behavior is sufficiently advanced to enable its applications in computer science and other practical domains. The years since World War II have been the occasion for an immense wave of new understandings and new techniques in which man has come to be viewed as an active processor of information. In the last decade or so, these understandings and techniques have engulfed the main areas of human experimental psychology[1]: perception,[2] performance,[3] memory,[4] learning,[5] problem solving,[6] psycholinguistics.[7] By now, cognitive psychology has come to be dominated by the information-processing viewpoint.

A major advance in understanding and technique brings with it, after some delay, an associated wave of applications for the new knowledge. Such a wave is about to break in psychology. The information-processing view will lead to a surge of new ways for making psychology relevant to our human needs. Already the concepts of information-processing psychology have been applied to legal eyewitness testimony[8] and to the design of intelligence tests.[9] And in the study of man-machine systems and engineering psychology, it has for some time been common to include a block diagram of the overall human information-processing system in the introductory chapter of textbooks,[10] even though the reach of that block diagram into the text proper is still tenuous. There are already the beginnings of a subfield, for which various names (associating the topic in different ways) have been suggested: user sciences,[11] artificial psycholinguistics,[12] cognitive ergonomics,[13] software psychology,[14] user psychology,[15] and cognitive engineering.[16]

[1] For representative examples see Lindsay and Norman's (1977) *Human Information Processing*, Anderson's (1980) *Cognitive Psychology and its Implications*, the *Handbook of Learning and Cognitive Processes* (Estes, ed. 1975-1978), the *Attention and Performance* collections of papers (Kornblum, 1973; Rabbitt and Dornič, 1975; Dornič, 1977; Requin, 1978; Long and Baddeley, 1981), and the journal *Cognitive Psychology.*

[2] Examples: Broadbent (1958), *Perception and Communication*; Green and Swets (1966), *Signal Detection Theory and Psychophysics*; Neisser (1967), *Cognitive Psychology*; Cornsweet (1970) *Visual Perception.*

[3] Examples: Fitts and Posner (1967), *Human Performance*; Welford (1968), *Fundamentals of Skill*; Kintsch (1974), *The Representation of Meaning in Memory*; Tversky (1977), "Feature of similarity"; Posner (1978), *Chronometric Explorations of the Mind.*

[4] Examples: Anderson and Bower (1973), *Human Associative Memory*; Baddeley (1976), *The Psychology of Memory*; Crowder (1976), *Principles of Learning and Memory*; Murdock (1974), *Human Memory, Theory and Data.*

[5] Examples: Fitts (1964), "Perceptual-motor skill learning"; Klahr and Wallace (1976), *Cognitive Development: An Information-Processing View*; Anderson (1981a), *Cognitive Skills and their Acquisition.*

[6] Example: Newell and Simon (1972), *Human Problem Solving.*

Our own goal is to help create this wave of application: to help create an applied information-processing psychology. As with all applied science, this can only be done by working within some specific domain of application. For us, this domain is the human-computer interface. The application is no offhand choice for us, nor is the application dictated solely by its extrinsic importance. There is nothing that drives fundamental theory better than a good applied problem, and the cognitive engineering of the human-computer interface has all the markings of such a problem, both substantively and methodologically. Society is in the midst of transforming itself to use the power of computers throughout its entire fabric—wherever information is used—and that transformation depends critically on the quality of human-computer interaction. Moreover, the problem appears to have the right mixture of industrial application and symbol manipulation to make it a "real-world" problem and yet be within reasonable reach of an extended cognitive psychology. In addition, we have personal disciplinary commitments to computer science as well as to psychology.

This book reports on a program of research directed towards understanding human-computer interaction, with special reference to text-editing systems. The program was undertaken as an initial step towards the applied information-processing psychology we seek. Before outlining individual studies, it is appropriate to sketch how this effort fits in with the larger endeavor.

[7] Example: Clark and Clark (1976), *Psychology and Language: An Introduction to Psycholinguistics.*

[8] Loftus (1979).

[9] Hunt, Frost, and Lunneborg (1973).

[10] Sheridan and Ferrell (1974); McCormick (1976).

[11] Vallee (1976).

[12] Sime and Green (1974).

[13] Sime, Fitter, and Green (1975).

[14] Shneiderman (1980).

[15] Moran (1981*a*).

[16] Norman (1980).

1.1. THE HUMAN-COMPUTER INTERFACE

The human-computer interface is easy to find in a gross way—jus follow a data path outward from the computer's central processor unti you stumble across a human being (Figure 1.1). Identifying its bound aries is a little more subtle. The key notion, perhaps, is that the user anc the computer engage in a communicative dialogue whose purpose is th accomplishment of some task. It can be termed a dialogue because bot the computer and the user have access to the stream of symbols flowin back and forth to accomplish the communication; each can interrupt query, and correct the communication at various points in the process All the mechanisms used in this dialogue constitute the interface: th physical devices, such as keyboards and displays, as well as computer' programs for controlling the interaction.

At any point in the history of computer technology there seems to b a prototypical user interface. A few years ago it was the teletypewriter currently it is the alphanumeric video-terminal. But the actual diversit is now much greater. All so-called "remote entry" devices count a interfaces; and a large number of such specialized devices exist in th commercial and industrial world to record sales, maintain inventor records, or control industrial processes. Almost all such devices ar fashioned from the same basic sorts of components (keyboards, button video displays, printers) and connect to the same sorts of information processing mechanisms (disks, channels, interrupt service routines).

The very existence of the direct human-computer interface is itself a emergent event in the development of computers. If we go back twent years, the dominant scheme for entering information into a compute consisted of a trio of people. First there was the user, someone wh wanted to accomplish some task with the aid of the computer. The use encoded what he wanted onto a coding sheet, then sent it to a secon person, the keypunch operator, who used an off-line device, th keypunch, to create a deck of punched cards that encoded the sam information in a different form. The cards in turn went to a thir person, the computer-operator, who entered the cards into the compute via the card reader. The computer then responded by printing message and data on paper for the operator to gather up and send back to th user. The relationship between the user and the computer was su ficiently remote that it should be likened more to a literar correspondence than to a conversational dialogue. It is the genera

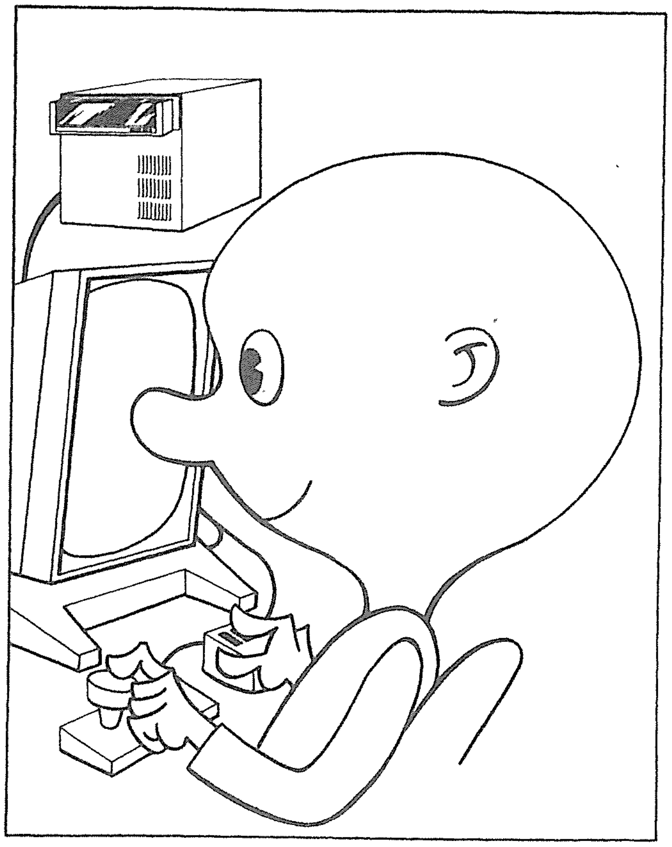

Figure 1.1. The human-computer interface.

demise of such arrangements involving human intermediaries, and the resultant coupling of the user directly to the computer, that has given rise to the contemporary human-computer interface. Whatever continued evolution the interface takes—and it will be substantial—human-computer interaction is unlikely ever to lose this character of a conversational dialogue.

Of course, there is much more to improving computer interfaces than simply making them conversational. Informal evidence from the direct experience of users provides numerous examples of current interface deficiencies:

> In one text-editing system, typing the word *edit* while in command mode would cause the system to select every-

thing, delete everything, and then insert the letter t (this last making it impossible to use the system Undo command to recover the deleted text because only the last command could be undone).

In another text-editing system, so many short commands were defined that almost any typing error would cause some disaster to happen. For example, accidentally typing CONTROL-E would cause the printer to be captured by the user. Since no indication of this event was given, no other users would be permitted to print until the other users eventually discovered who had the printer. In an even more spectacular instance, accidentally typing CONTROL-Z would delete all the user's files—permanently.

In one interactive programming system, misspelling a variable name containing hyphens (a common way of marking off parts of a name) would cause the system to rewrite the user's program, inserting code to subtract the parts of the name. In many cases, the user would have to mend his program by hand, laboriously searching for and editing the damaged code.

In a set of different subsystems meant to be used together, the name "List" was given to many different commands, each having a different meaning: (1) send a file to the printer to make a hardcopy, (2) show the directory of files on the display, (3) show the content of a file on the display, (4) copy the workspace to a file, (5) create a particular kind of data structure.

Yet, when one looks at the teletype interfaces of yesterday, it is cle that substantial progress has been made. The emergence of the dire human interface, circumventing the keypuncher and operator, must its be counted as an improvement of enormous value. We now ha interfaces that allow the use of computers for such highly interactive tas as making engineering drawings and taking airline reservations. B despite considerable advancements, the systems we have are often ragg and in places are sufficiently poor to cripple whole ranges of us

What strikes one most noticeably about existing interfaces, besides all the little ways they fail, is that their failures appear to be unnecessary. Why, when interaction could be so smooth, even elegant, is it often so rough, even hazardous? Two observations may help explain this perplexing state of affairs.

First, interaction with computers is just emerging as a human activity. Prior styles of interaction between people and machines—such as driver and automobile, secretary and typewriter, or operator and control room—are all extremely lean: there is a limited range of tasks to be accomplished and a narrow range of means (wheels, levers, and knobs) for accomplishing them. The notion of the *operator* of a machine arose out of this context. But the user is not an operator. He does not operate the computer, he communicates with it to accomplish a task. Thus, we are creating a new arena of human action: communication *with* machines rather than operation *of* machines. What the nature of this arena is like we hardly yet know. We must expect the first systems that explore the arena to be fragmentary and uneven.

Second, the radical increase in both the computer's power and its performance/cost ratio has meant that an increasing amount of computational resources have become available to be spent on the human-computer interface itself, rather than on purely computational tasks. This increase of deployable resources exacerbates the novelty of the area, since entirely new styles of interaction become available coincidentally with an increased amount of computational ability available per interaction. These new styles often lead to completely new interfaces, which are then even more ragged than before. At the same time, opportunities for the invention of good interfaces also increase rapidly, accounting for the leaps and bounds we have seen in terms of major improvements in functionality and ease of use.

1.2. THE ROLE OF PSYCHOLOGY

Many in the computer field agree that there is an obvious way to design better human-computer interfaces. Unfortunately, they disagree on what it is. It is obvious to some that psychological knowledge should be applied. Their slogan might be, in the words of Hansen (1971): "Know the user!" It is obvious to others that the interface should simply be designed with more care—that if designers were given the goal of good interfaces, rather than stringent cost limits or tight deadlines, then they would produce good designs. Their slogan might be: "Designers are users too—just give them the time and freedom to design it right!" And it is obvious to others still that one should pour the effort into some new components—flat displays, color graphics, or dynamically codeable microprocessors in the terminal. Their slogan might be: "Make the components good enough and the system will take care of itself!"

Who is to gainsay each of these their point? The technology limits, often severely, what can be done. All the human engineering in the world will not turn a 10-character-per-second teletypewriter into a high-resolution graphics terminal. The history of terminal development so far is writ largely in terms of advances in basic interface components, most notably the resources to allow substantial computational cycles to be devoted to the interface. It is easy to point to current limitations whose lifting will improve the interface by orders of magnitude. Immense gains will occur when the display holds not the common 24 × 80 characters (the typical alphanumeric video terminal, widely available today), but a full page of 64 × 120 characters (the typical 1000 × 800 pixel video terminal, available at a few places today), or even the full drafting board of 512 × 512 characters (not really available anywhere, yet, as far as we know).

Moreover, any accounting will have to credit the majority of the capabilities and advances at the interface to design engineers and only a few of them to psychologists. However many imperfections there remain in the interface, the basic capabilities and inspired creations that do exist came out of an engineering analysis of the functions needed and the fact that the designer, being human, could empathize directly with the user.

And yet, there remain the mini-horror stories—of systems where, after the fact, it became clear that either the nature or the limitations of the user were not appreciated, and some design foolishness was committed. Since it is these stories that come to mind in discussing the role of the human at the interface, it is often assumed that all that one needs are ways of checking to be sure that the obvious is not overlooked; "All we need from psychology is a few good checklists!" might be the slogan here. But as we shall see, there is more to human-computer interaction than can be caught with checklists.

The role psychology might be expected to play in the design of the user-computer interface is suggested by the results it was able to achieve for military equipment during World War II. At that time, it had become apparent that a strong limiting factor in realizing the potential of man-machine systems, such as radar sets and military aircraft, lay in the difficulty of operating the equipment. Out of a wartime collaboration between natural scientists, engineers, and psychologists came major advances, not only with respect to the man-machine systems being designed, but also with respect to psychological theory itself. Examples of the latter include the theory of signal detection, manual control theory, and a methodology for the design of cockpit instrument displays. That with psychological attention to human performance airplanes became more flyable encourages us to believe that with psychological attention to human performance computers can become more usable.

1.3. THE FORM OF AN APPLIED PSYCHOLOGY

What might an applied information-processing psychology of human-computer interfaces be like and how might it be used? Imagine the following scenario:

> A system designer, the head of a small team writing the specifications for a desktop calendar-scheduling system, is choosing between having users type a key for each command and having them point to a menu with a lightpen. On his whiteboard, he lists some representative tasks users of his system must perform. In two columns, he writes the steps needed by the "key-command" and "menu" options. From a handbook, he culls the times for each step, adding the step times to get total task times. The key-command system takes less time, but only slightly. But, applying the analysis from another section of the handbook, he calculates that the menu system will be faster to learn; in fact, it will be learnable in half the time. He has estimated previously that an effective menu system will require a more expensive processor: 20% more memory, 100% more microcode memory, and a more expensive display. Is the extra expenditure worthwhile? A few more minutes of calculation and he realizes the startling fact that, for the manufacturing quantities anticipated, training costs

for the key-command system will exceed unit manufacturing costs! The increase in hardware costs would be much more than balanced by the decrease in training costs, even before considering the increase in market that can be expected for a more easily learned system. Are there advantages to the key-command system in other areas, which need to be balanced? He proceeds with other analyses, considering the load on the user's memory, the potential for user errors, and the likelihood of fatigue. In the next room, the Pascal compiler hums idly, unused, awaiting his decision.

The system designer is engaged in a sort of psychological civ engineering, trading computed parameters of human performance agains cost and other engineering variables. The psychological science bas necessary to make possible his design efforts is the sort of applie psychology that is the topic of this book. Such a psychology mus necessarily be homogeneous in form with the rest of the engineerin science base to allow tradeoffs between psychological and other desig considerations. To be useful, we would argue, such a psychology mus be based on task analysis, calculation, and approximation.

Task Analysis. When psychology is applied in the context of specific task, much of the activity hardly seems like psychology at al but rather like an analysis of the task itself. The reason for this is clear humans behave in a goal-oriented way. Within their limited perceptua and information-processing abilities, they attempt to adapt to the tas environment to attain their goals. Once the goals are known or can b assumed, the structure of the task environment provides a large amoun of the predictive content of psychology.

Calculation. The ability to do calculations is the heart of useful engineering-oriented applied science. Without it, one is crippled. Applications are, of course, still possible, as witness mental testing, behavio modification, assertiveness training, and human-factors investi-gations o display readability. But what is needed to support an engineerin analysis are laws of parametric variation, applicable on the basis of a tasl analysis.

Psychology is not strong on calculation, though a few useful laws such as Power Law of Practice, exist. The reason might be thought t be an inherent characteristic of psychology, or maybe even mor generally, of all human sciences. Our view is the opposite. Psycholog is largely non-calculational because it has followed a different drummer. It has been excessively concerned with hypothesis testing—with building techniques to discriminate which of two ideas is right. If one changes what one wants from the science, one will find the requisite techniques. Interestingly, a branch of the human sciences, work-measurement industrial engineering, indeed asked a different question—namely, how long would it take people to do preset physical tasks—and it obtained useful answers.

Approximation. If calculations are going to be made rapidly, they are necessarily going to be over-simplified. Nature—especially human nature—is too complex to be written out on the back of an envelope. But in engineering, approximations are of the essence. It is vital to get an answer good enough to dictate the design choice; additional accuracy is gilding the computational lily.

Again, psychology has in general not asked after approximations, though it has certainly learned to talk in terms of simplified models. The neglect of approximation has been especially encouraged by the emphasis on statistical significance rather than on the magnitude of an effect. A difference of a few percent in performance at two levels of an independent variable is usually of little practical importance and can often be ignored in an approximation, even if the difference is highly significant statistically. But if there is no external criterion—no design decision to be made, for instance—then there is no way to tell which approximations are sufficient.

But, whereas an applied psychology of human-computer interaction should be characterized by task analysis, calculation, and approximation, these are not the only considerations. It is obvious that an applied psychology intended to support cognitive engineering should also be relevant to design. It is less obvious, but nonetheless true, that to be successful, an applied psychology should be theory-based.

RELEVANT TO DESIGN

Design is where the action is in the human-computer interface. It is during design that there are enough degrees of freedom to make a difference. An applied psychology brought to bear at some other point is destined to be half crippled in its impact.

We suspect that many psychologists would tend to pick evaluation as the main focus for application (though some might have picked training). Evaluation is what human factors has done best. Given a real system, one can produce a judgment by experimentation. Thus, the main tool in the human-factors kit has been the methodology of experimental design, supported by concomitant skill in experimental control and in statistics with which to assess the results. The emphasis on evaluation is widespread: There is a whole subfield of psychology whose concern is to evaluate social action programs. The testing movement is fundamentally evaluational in character, whether concerned with intelligence testing or with clinical assessment.

Applying psychology to the evaluation of systems is assuredly easier than applying it to the design of systems. In evaluation, the system is given; all its parts and properties are specified. In design, the system is still largely hypothetical; it is a class of systems. On the other hand, there is much less leverage in system evaluation than in system design. In design, one wants results expressed explicitly as a function of some controllable parameters, in order to explore optimization and sensitivity. In evaluation, this urge is much diminished; experimental evaluation is so expensive as to be prohibitive, permitting exploration of only two or three levels of each independent variable. Most importantly, by the time a system is running well enough to evaluate, it is almost inevitably too late to change it much. Thus, an applied psychology aimed exclusively at evaluation is doomed to have little impact.

There are several choices for how to institutionalize an applied psychology. First, psychologists could be the primary professionals in the field. Though possible in some fields, such as mental health, counseling, or education, we think this arrangement unlikely for computers. The field is already solely in the possession of computer engineers and scientists. Second, psychologists could be specialists, either as members of separate human-factors units within the organizations or as another individual specialty within the primary design team. Our reasons for not favoring separate psychology units reflect the additional separation we believe they imply between the psychology and the development of interfaces. Application of psychology would shift too strongly towards evaluation and away from the main design processes.

We favor a third choice: that the primary professionals—the computer system designers—be the main agents to apply psychology. Much as a civil engineer learns to apply for himself the relevant physics of bridges, the system designer should become the possessor of the relevant applied psychology of human-computer interfaces. Then and only then will it become possible for him to trade human behavioral considerations against the many other technical design considerations of system configuration and implementation. For this to be possible, it is necessary that a psychology of interface design be cast in terms homogeneous with those commonly used in other parts of computer science and that it be packaged in handbooks that make its application easy. Thus, the system designer in our scenario finds the design handbook more efficient to use than plunging blindly into code with his Pascal compiler, although he may still find it profitable to engage in exploratory implementation.

THEORY-BASED

An applied psychology that is theory-based, in the sense of articulating a mechanism underlying the observed phenomena, has advantages of insight and integration over a purely empirical approach. The point can be made by reference to two examples of behavioral science lacking a strong theory in this sense: work-study industrial engineering, referred to earlier, and intelligence testing. Rather than develop the theory of skilled movement, the developers of the several movement time systems chose an empirical approach, tabulating the times to make various classes of movements and ignoring promising theoretical developments such as Fitts's Law (at least until recently). Although their tables of motion times ran to four significant figures, they ignored the variance of the times and interactions between sequential motions, thus rendering the apparent precision illusory. This lack of adequate theoretical development made the work, despite its impressive successes, vulnerable to attacks from outside the field (see Abruzzi, 1956; Schmidtke and Stier, 1961). Similarly, in mental testing, the lack of a psychological theory of the mental mechanisms underlying intelligence (as opposed to a purely statistical theory of test construction) has put the validity of mental tests in doubt despite, again, impressive successes.

It is natural for an applied psychology of human-computer interaction to be based theoretically on information-processing psychology, with the latter's emphasis on mental mechanism. The use of models in which man is viewed as a processor of information also provides a common framework in which models of memory, problem solving, perception, and behavior all can be integrated with one another. Since the system designer also does his work in information-processing terms, the emphasis is doubly appropriate. The lack of this common framework is one reason why it would be difficult to meld in important techniques such as the use of Skinnerian contingent reinforcement. It is not that the techniques are not useful in general, nor that they cannot be applied to the problems of the human-computer interface; but within the framework that underlies this book, they would show up as isolated techniques.

The psychology of the human-computer interface is generally individual psychology: the study of a human behaving within a non-human environment (though, interestingly, interacting with another active agent). But within the study all psychological functioning is included—motor, perceptual, and cognitive. Whereas much psychology tends to focus on small micro-tasks studied in isolation, an applied psychology must dwell on the way in which all the components of the human processor are integrated over time to do useful tasks. For example, it might take into account interactions among the following: the ease with which commands can be remembered, the type font of characters as it affects legibility of the commands, the number of commands in a list, and anything else relevant to the particular interface. The general desirability of such wide coverage has never been in doubt. It appears in our vision of an applied psychology because wide coverage, especially the incorporation of cognition, now seems much more credible than it did twenty years ago. On the other hand, motivational and personality issues are not included. Again, there is hardly any doubt of the desirability of including them in an applied psychology, but it is unclear how to integrate the relevant existing knowledge of these topics.

1.4. THE YIELD FOR COGNITIVE PSYCHOLOGY

The textbook view is that as a science develops it sprouts applications, that knowledge flows from the pure to the applied, that the backflow is the satisfaction (and support) that comes to a science from benefiting society. We have been reminded often enough that such a view does violence to the realities in several ways. Applied domains have a life and source of their own, so that many ingenious applications do not spring from basic science, but from direct understanding of the task in an applied context—from craft and experience. More importantly in the present context, applied investigations vitalize the basic science; they reveal new phenomena and set forth clearly what it is that needs explanation. The mechanical equivalent for heat, for instance, arose from Count Rumford's applied investigations into the boring of brass cannon; and the bacteriological origin of common infectious diseases eventually arose, in part, out of studies by Pasteur on problems besetting the fermentation of wine. The basic argument was made for psychology by Bryan and Harter (1898); and numerous applied psychological models exist to remind us of what is possible (for example, Bryan and Harter's 1898 and 1899 studies of telegraphy, Book's 1908 studies of typewriting, and Dansereau's 1968 study of mental arithmetic).

These general points certainly hold for an applied cognitive psychology, and on the same general ground that they hold for all sciences. However, it is worth detailing the three main yields for cognitive psychology that can flow from a robust applied cognitive psychology.

The first contribution is to the substance of basic cognitive psychology. The information-processing revolution in cognitive psychology is just beginning. Many domains of cognitive activity have hardly been explored. Such explorations are not peripheral to the basic science. It is a major challenge to the information-processing view to be able to explain how knowledge and skill are organized to cope with all kinds of complex human activities. Each application area in fact becomes an arena in which new problems for the basic science can arise. Each application area successfully mastered offers lessons about the ways in which the basic science can be extended to cover new areas. Ultimately, as a theory becomes solidified, application areas contribute less and less to the basic science. But at the beginning, just the reverse is true.

The domain of human-computer interaction is an example of such an unexplored domain. It has strong skill components. People who interact with computers extensively build up a repertoire of efficient, smooth, learned behaviors for carrying out their routine communicative activities. Yet, the interaction is also intensely cognitive. The skills are wielded within a problem-solving context, and the skills themselves involve the processing of symbolic information. As we shall see in abundance, even the most routine of these activities, such as using a computer text-editing program, requires the interpretation of instructions, the formulation of sequences of commands, and the communication of these commands to the computer.

The second contribution is to the style of cognitive psychology rather than to its substance. We believe that the form of the psychology of human-computer interaction, with its emphasis on task analysis, calculation, and approximation, is also appropriate for basic cognitive psychology. The existing emphasis in psychology on discriminating between theories is certainly understandable as a historical development.

However, it stifles the growth of adequate theory and of the cumulation of knowledge by focusing the attention of the field on the consequences of theories, however uninteresting in themselves, that can be used to tell whether idea A or idea B is correct. Measurements come to have little value in themselves as a continually growing body of useful quantitative knowledge of the phenomena. They are seen instead primarily as indicators fashioned to fit the demands of each experimental test. Since there is no numerical correspondence across paradigms in what is measured, the emphasis on discrimination fosters a tendency towards isolation of phenomena in specific experimental paradigms.

The third contribution is simply that of being a successful application, though it sounds a bit odd to say it that way. Modern cognitive psychology has been developing now for 25 years. If information-processing psychology represents a successful advance of some magnitude, then ultimately it must both affect the areas in which psychology is now applied and generate new areas of application.

1.5. THE YIELD FOR COMPUTER SCIENCE

It is our strong belief that the psychological phenomena surrounding computer systems should be part of computer science. Thus, we see this book not just as a book in applied psychology, but as a book in computer science as well. When university curriculum committees draw up a list of "what every computer scientist should know to call himself a computer scientist," we think models of the human user have a place alongside models of compilers and language interpreters.

The fundamental argument is worth stating: Certain central aspects of computers are as much a function of the nature of human beings as of the nature of the computers themselves. The relevance of both computer science and psychology to the design of programming languages and the interface is easy to argue, but psychological considerations enter into more topics in computer science than is usually realized. The presumption that has governed two generations of operating systems, for instance, that time-sharing systems should degrade response time as the number of users increases, is neither dictated by technology nor independent of the psychology of the user. A sufficiently crisp model of the effects of such a feature on the user could have turned the course of development of operating systems into quite different channels of development (into the logic of guaranteed service, contracted service, or proportionately graded services, for example). The yield for computer science that can flow from an applied psychology of human-computer interaction is engineering methods for taking the properties of users into account during system design.

1.6. PREVIEW

In this book, we report on a series of studies undertaken to understand the performance of users on interactive computing systems. Since new knowledge and insight are often achieved by first focusing on concrete cases and then generalizing, we direct a major portion of our effort towards user performance on computer text-editing systems. From this beginning, we try to generalize to other systems and to cognitive skill generally. We address four basic questions: (1) How can the science base be built up for supporting the design of human-computer interfaces? (2) What are user performance characteristics in a specific human-computer interaction task domain, text-editing? (3) How can our results be cast as practical models to aid in design? (4) What generalizations arise from the specific studies, models, and applications?

SCIENCE BASE

Chapter 2 begins by discussing the existing scientific base on which to erect an applied psychology of the human-computer interface. It does not review all the sources in their own terms—what is available from cognitive psychology, human factors, industrial engineering, manual control, or the classical study of motor skills—rather, it lays out a model of the human information-processor that is suited to an applied psychology and justified by current research.

TEXT-EDITING

Attention then turns to a detailed examination of text-editing as a prototypical example of human-computer interaction. An elementary requirement for understanding behavior at the interface is some gross quantitative information about user behavior, to provide a background picture against which to place more detailed studies in context. The three studies in Chapters 3 and 4 provide such a picture. Two of these (Chapter 3), a benchmark study comparing text-editing systems and a study of the individual user differences, allow one to assess the variability in performance time arising from editing system design and from individual user differences. The third study (Chapter 4) uses the data of Chapter 3 to explore how well a simple model, in which all editing modifications are assumed to take the same time, does at analyzing tradeoffs between using a computer text-editor vs. using a typewriter.

The next three chapters develop an information-processing model for the behavior of users with an editing system. Chapter 5 introduces the basic theory. The user is taken to employ goals, operators, methods, and selection rules for the methods (the GOMS analysis) to accomplish an editing task from a marked-up manuscript. Experimental verification of the analysis is given, and the effect on accuracy due to the detail with which the analysis is applied is also investigated. The routine use of an editing system is discussed as an instance of cognitive skill. Chapter 6 extends the model in three ways. First, the model is reduced to a complete, running computer simulation of user performance. Second, the analysis is extended to user behavior on a display-oriented system. Third, stochastic elements are introduced into the model to predict the distributions of performance times. Chapter 7 examines in detail one suboperation of editing: selecting a piece of text. Four different devices for doing this are tested, and a theoretical account is given for their performance.

ENGINEERING MODELS

Chapters 8 and 9 focus on the ways in which the GOMS analysis can be simplified to provide practical models for predicting the amount of time required by a user to do a task. In Chapter 8, a model at the level of individual keystrokes is presented that is sufficiently simple and accurate to be a design tool. The model is validated over several systems, tasks, and users; and examples are given for ways in which the model could be used in engineering applications. In Chapter 9, a second simplification of the GOMS analysis, this time at a more gross level, is presented This model is suited for cases where, as in the early stages of design, the system to be analyzed is not fully specified.

EXTENSIONS AND GENERALIZATIONS

So far, the studies have focused mostly on manuscript editing and on similar tasks where the user carries out a set of instructions. Chapter 10 extends the same kind of analysis to a particular problem-solving activity:

the use of a computer system to lay out a VLSI electronic circuit. The analysis shows that the user behavior exhibits many of the characteristics of manuscript editing and that the behavior is indeed a routine cognitive skill, partially understandable in terms of the concepts already introduced.

Chapter 11 attempts to place results from the above studies in a larger theoretical context. It continues the discussion of text-editing as an instance of cognitive skill and the relationship between cognitive skill generally and problem solving. Chapter 12 addresses the role of psychological studies in design. It is argued that psychological studies should emphasize the creation of performance models. The several methods of doing this are discussed and provide a framework for summarizing the thrust of the present book. A number of guidelines for systems development that arise from our studies are listed.

12. Applying Psychology to Design

In this chapter, we return to the theme of an *applied* psychology introduced in Chapter 1 and attempt to tie our studies together from the vantage point of how such knowledge might be used in design. We do this first by presenting a framework showing the way in which psychological results can be applied to design and then by mapping our studies into this framework. We also summarize an application-oriented extension of our work by Roberts. Finally, we list some general system design principles suggested by the studies.

12.1. A FRAMEWORK FOR APPLYING PSYCHOLOGY

In Chapter 1 we proposed that an applied psychology of human-computer interaction should be relevant to the system design process itself (not just to after-the-fact evaluation) and that the designer himself should do the actual application. Such an applied science must be based on information-processing models, whose applicability to design depends on three critical features: task analysis, calculation, and approximation. This proposal rests on a view (until now implicit) of how psychology can be applied to system design. We now present our view by briefly sketching a framework for application. This framework includes (1) the structure and performance of the human-computer system, (2) performance models for predicting the performance of the human-computer system, and (3) design functions for using the performance models in the design process.

THE HUMAN-COMPUTER SYSTEM

Our present object of study is the *human-computer system*, which consists of a human user interacting with a computer to accomplish a task. The user, the computer, and the task are the structural components of the system. Human-computer systems vary in many different respects, called *structural variables*, in each of the components. Systems address different task domains, and they have different models of the tasks in any given domain. Users vary widely in general intellectual ability, experience with computers, specific knowledge of the task, specific knowledge of the computer, cognitive style, and perceptual-motor skills. User-interface aspects of computers vary in system architecture, dialogue style, command syntax, input devices, and so on.[1] The combination of all these variables produces a vast space of possible human-computer systems.

The ultimate concern of an applied psychology is not so much with the structure of the human-computer system per se, as with its *performance*. There are many different aspects to performance, which we call *performance variables*. The basic performance variables of a human-computer system are concerned with what tasks the system can do (functionality), how long it takes to acquire the functionality (learning), how long it takes to accomplish tasks (time), how frequently errors occur and how consequential they are, how well tasks are done (quality), and how robust the system is in the face of unexpected conditions. Other performance measures are possible, such as performance under extreme conditions (fatigue and stress) and the performance demands on the user's memories (Working Memory and Long-Term Memory). Finally, there are variables concerning the user's subjective feeling about the system. All these performance variables are potential areas of concern to the system designer.

The performance variables of a human-computer system are determined by its structural variables. This can be summarized in a formula analogous to the Rationality Principle (Chapter 2):

$$\textit{Task} + \textit{User} + \textit{Computer} \rightarrow \textit{System Performance}. \qquad (12.1)$$

[1] A systematic analysis of the structure of the user-interface aspects of interactive computer systems is a difficult undertaking. For some attempts, see Moran (1981*a*); Young (1981); Newman and Sproull (1979, Ch. 28); and Ramsey and Atwood (1979).

It is the task of an applied psychology to discover the specific relationships between the structural and performance variables of human-computer systems.

PERFORMANCE MODELS

The design of a human-computer system begins with a set of requirements, which includes both structural constraints and performance goals. The designer's job is to specify a human-computer system satisfying the requirements. But while specifications of a system are readily checked against the structural constraints, the performance aspects of a system are not derivable from a descriptive specification. A special kind of representation of the human-computer system is needed for this, which we call a *performance model.* To predict the performance of a system, the designer must construct a specific performance model from the system's structural specifications and then use the model to generate a prediction:

$$\begin{aligned}&\mathit{Model}\,(\mathit{Task},\ \mathit{User},\ \mathit{Computer})\\ &\qquad\qquad \rightarrow\ \mathit{Performance\ Prediction}\,. \end{aligned} \tag{12.2}$$

The concept of a performance model is the key notion in this framework. It is useful to construe this notion functionally, i.e., as any model or description that can be used to predict system performance. Performance models can be roughly categorized as experimental models, symbolic models, and database models. Experimental models consist of actual human users with actual running programs or physical mock-ups. Such models are *run*, and performance variables are *measured.* Symbolic models are calculational, algebraic, or simulation models. They are represented on paper or in a computer and have no actual human component (although, of course, they model the user). Performance values are obtained by *computation* (by hand or computer). Database models are stores of pre-measured or pre-calculated data. Performance values are obtained simply by *look-up.* Each of these different kinds of performance models has its place in the system design process.

DESIGN FUNCTIONS

The predictive function of performance models is primarily *evaluative*: given a structure, predict performance. The designer's problem, however, is *generative*: given performance requirements, design the structure.

Although it is possible to invert, partially at least, some performance models to generate design ideas, it is not possible to invert Formula 12.2 in a general way. For any interesting real-world domain of design, there cannot be any global synthesis function that maps requirements into a structure. How, then, can performance models be useful in design? To answer this question we must consider the nature of the design process.

Design, as all designers know, is not a simple top-down or bottom-up process of synthesizing a design solution from requirements. Design is an open process, in the sense that the design problem is constantly being redefined. Many requirements can emerge only in the course of the design process, when partial design solutions provide enough context to realize which issues are really important. Thus, design proceeds in a complex, iterative fashion in which various parts of the design are incrementally generated, evaluated, and integrated. At the risk of being over-simplistic, we characterize the complex process of design as consisting of a set of different kinds of *design functions*, each attending to a specific design subproblem:

$$\textit{Design Process} = \textit{a set of Design Functions}. \qquad (12.3)$$

Although we do not pretend to have even a crude taxonomy of design functions, we can list some examples to make the notion more concrete. We can group design functions into three broad categories: evaluation, parametric design, and structural design. Evaluation, as we just noted, refers to the situation in which the structure of the system (or of part of the system) has been specified and its performance needs to be understood. Parametric design refers to the situation in which the structure of the system is relatively fixed and there are a set of quantitative parameters of the structure to be determined. (What makes parametric design tractable for analysis is the assumption that the remaining structure of the system will not change in the range of parameter values under consideration.) Structural design is where a part of the system is configured or restructured to satisfy specific requirements. There are several functions in structural design, such as to identify an opportunity for a change, to diagnose a problem, to generate an improvement, and to synthesize a new structure.

Design functions require the use of performance models to solve particular design subproblems:

$$\textit{Design Function}\,(\textit{Design Subproblem, Model}) \rightarrow \textit{Solution}.$$

The dependence on performance models is clear in evaluation and parametric design. Formal performance models are seldom used in structural design, although there are usually implicit, informal performance assumptions underlying the design functions, which can be viewed as vague, informal performance models. But partial inversions of more formal performance models can also be used (e.g., to diagnose the causes of performance deficits).[2]

This framework for applying psychology to design emphasizes the pivotal role of performance models. Without models, the designer cannot predict the performance of the system he is designing. If he cannot predict performance, he will not be able to come to grips with performance requirements. And if he does not deal with performance requirements, then other requirements will dominate the design of the system—and the user will be neglected. Thus, in our view it is clear that *the role of an applied psychology is to supply performance models for the designer.*

12.2. CONTRIBUTIONS TO APPLICATION

We now have a framework for considering how the studies in this book address the issues of applying psychology to system design. We proceed by enumerating the principal aspects of the application framework—the human-computer system, performance models, and design functions—and by showing how far we have progressed and where needs exist for future research.

THE HUMAN-COMPUTER SYSTEM

According to Formula 12.1, the structural variables of the human-computer system—the task, the user, and the computer—determine its performance variables. Figure 12.1 lists a set of structural variables for characterizing the variety of possible human-computer systems, and

[2] Some of the design principles to be presented in Section 12.4 can be viewed as inversions of our models of cognitive skill in human-computer interaction. For example, Principle 8 takes the GOMS model of method-selection in Chapter 5 and, instead of using the model to predict performance, observes that performance will be better if method alternatives are designed so they can be selected with a simple set of method-selection rules.

Structural Variables	Studies (Chapters/Sections)
TASK VARIABLES	
Task domain	Text-editing (3-6, 8) Graphics (8) Page layout (9) Circuit design (10)
Task model	POET editing analysis (5.1) BRAVO editing analysis (6.1)
USER VARIABLES	
INTELLECTUAL ABILITIES	
General intelligence	———
Technical ability	Individual differences (3.3)
COGNITIVE STYLE	
Risk preference	Selection rules (5.2) Error rates (12.3)
Curiosity	———
Persistence	———
EXPERIENCE	
Experience on system	Individual differences (3.3)
Frequency of system use	Individual differences (3.3)
KNOWLEDGE	
Method knowledge	Selection rules (5.2)
Conceptual knowledge	———
Task expertise	———
PERCEPTUAL-MOTOR SKILL	
Typing rate	Individual differences (3.3) Model validation (8.4)
Manual skill	———
COMPUTER VARIABLES	
Dialogue style	Compare editors (3.2, 12.3) Editor vs typewriter (4) Interactive systems (8.3)
Command syntax	Interactive systems (8.3)
Naming conventions	———
Display layout	———
Input devices	Pointing devices (7)
Response time	———

Figure 12.1. Studies classified by structural variables.

Performance Variables		Studies (Chapters/Sections)
BASIC PERFORMANCE MEASURES		
Functionality	What tasks can the user accomplish with the system?	Editing task population (12.3)
Learning	How does his performance improve over time?	Pointing devices (7.3) Text editing (12.3)
Time	How long does it take the user to do a task with the system?	Editing benchmarks (3.2) Individual differences (3.3) Editor vs. typewriter (4) POET editing (5) BRAVO editing (6) Pointing devices (7) Interactive systems (8) Page layout (9) Circuit design (10) Text editing (12.3)
Error	What errors are made, how frequently, and how consequential are they?	POET editing (5.4) Pointing devices (7.3) Circuit design (10.3) Text editing (12.3)
Quality	How good is the output?	——
Robustness	How does performance adapt to unexpected conditions or to new tasks?	——
SUBJECTIVE MEASURES		
Acceptability	How does the user subjectively rate the system?	——
Enjoyableness	How much fun is it to use?	——
EXTREME CONDITIONS		
Fatigue	How does performance degrade over time?	——
Stress	How does performance degrade under adverse conditions?	——
MEMORY VARIABLES		
WM Load	How much immediate information does the user have to keep in Working Memory?	——
LTM Recall	How easy is it for the user to recall information needed to accomplish a task?	——

Figure 12.2. Studies classified by performance variables.

Figure 12.2 lists a set of performance variables for characterizing the behavior of these systems.

Task Variables. Our strategy has been to focus on a single task domain and then to try to generalize to other domains. We have therefore been largely concerned with text-editing. We have generalized the results to other human-computer interaction task domains in the Keystroke-Level Model (Chapter 8), in the page-layout analysis (Chapter 9), and in the study of a circuit-layout system (Chapter 10). Within the domain of text-editing, we have analyzed two types of task models for text-editing—the line-structure model of text in POET and the two-dimensionally displayed character-stream model of text in BRAVO.

User Variables. We have not attempted to explore user variables systematically, except for the preliminary individual-differences study in Chapter 3. Instead, we have focused on expert users (who are best characterized by the knowledge and experience variables in Figure 12.1). Our strategy was to build a solid theoretical and empirical characterization of the expert user before attending to novice and casual users. However, we have seen some variations within experts, such as their knowledge of methods in the method selection study in Section 5.2.

Computer Variables. We have not attempted to explore computer variables systematically. Rather, our focus has been on how the user adapts to a given computer system structure. However, we have studied a variety of computer systems interfaces, from 1960's-style teletypewriter-oriented systems to state-of-the-art display-based systems. In some cases, we have directly compared behavior on alternative styles of system, such as in the studies of Chapters 3 and 8.

Performance Variables. Figure 12.2 clearly reveals our deliberate emphasis on performance time, which goes hand-in-hand with our emphasis on expert users. We have also presented a few modest accounts of the errors made by expert users. Our focus on performance time does not imply that we think the other performance variables are less important; indeed, they may be more important in many human-computer interaction contexts.

PERFORMANCE MODELS

Performance models, according to Formula 12.2, predict the performance of the human-computer system from a specification of its structure. Figure 12.3 lists several kinds of performance models. As can be seen in the figure, almost every kind of performance model has been

Performance Models		Studies (Chapters/Sections)
EXPERIMENTAL MODELS		
Running system	Use actual system.	Benchmark comparison (3.2) Editor evaluation (12.3)
Analogue system	Use another similar running system.	Layout test (9.2)
Mock-up system	Use a physical mock-up.	———
SYMBOLIC MODELS		
Calculational model	Code performance as a set of operations.	Manuscript editing (5) Keystroke-Level Model (8) Unit task analysis (9)
Simulation model	Code performance in a runable program.	BRAVO simulation (6)
Algebraic model	Represent relationships between variables and parameters as equations.	Editor vs. typewriter (4) Fitts's Law (7) New method analysis (8.4)
DATABASE MODELS		
Data table	Look up a pre-measured or pre-calculated value.	Model Human Processor (2) Benchmark data (3.2) Individual differences (3.3) Manuscript editing (5) Pointing devices (7) Keystroke-level operators (8) Editing data (12.3)
Checklist	Check design against principles or guidelines.	Design principles (12.4)

Figure 12.3. Studies classified by performance models.

presented. Our main emphasis, of course, has been on the development of symbolic models, especially calculational models: the GOMS family of models (Chapter 5), the Keystroke-Level Model (Chapter 8), and the Unit-Task-Level Model (Chapter 9). We have also presented a simulation model (Chapter 6) and several algebraic models, such as Fitts's Law (Chapter 7). We used running systems for the benchmark studies in Chapter 3. Finally, we have tabulated data that are useful databases (e.g., Figures 2.1, 2.2, 5.15, 7.4, 8.1, and 8.2).

DESIGN FUNCTIONS

The process of system design, according to Formula 12.3, consists of a set of design functions, which address design subproblems and which use performance models in finding solutions. We classified the design

Design Function		Studies (Chapters/Sections)
EVALUATION		
Compare systems	Compare on given performance variables.	Benchmark comparison (3.2) Device evaluations (7) Computed benchmark (8.4) Editor comparison (12.3)
Evaluate system	Compare against some standard.	Layout system (9) Editor evaluation (12.3)
PARAMETRIC DESIGN		
Optimize parameter	Find best value on given performance variables.	Editor vs. typewriter (4.2)
Analyze sensitivity	Relate parameter value to performance.	Editor vs. typewriter (4.4) New method analysis (8.4) Layout calculation (9.3)
STRUCTURAL DESIGN		
Identify opportunity	Find place where system can be improved.	Crossover point (4.2) Information rate limit (7.3) New method (8.4) Icarus Move command (10)
Diagnose problem	Pinpoint structural component causing problem.	Crossover point (4) New method (8.4)
Generate improvement	Find structural change.	New method (8.4)
Synthesize structure	Create new structure.	———

Figure 12.4. Studies classified by design functions.

functions as evaluation, parametric design, and structural design. Figure 12.4 lists several design functions, along with the studies illustrating them. The coverage is heaviest in evaluation and parametric design, where performance models are most clearly useful. System comparison usually involves experimentation (such as the benchmark study in Chapter 3 and the comparison of pointing devices in Chapter 7), but we have proposed the notion of a calculated benchmark (Section 8.4) for making comparisons analytically. The typewriter-versus-editor analysis in Chapter 4 illustrates both parameter optimization and sensitivity analysis. We have had less to say about structural design, especially the synthesis of a new design, for which we have no examples. Perhaps the best illustration of structural design functions is the analysis of alternative methods in Section 8.4, where an opportunity was identified (the task), the problem diagnosed (the awkwardness of the existing methods), and an improvement generated (the new method).

APPLICATION SUMMARY

The principal contribution of our studies to application is a set of specific performance models. This is consonant with the view, sketched in Section 12.1, that performance models are the keystones in the application of psychology to system design. The main limitations are that the models are restricted to predicting the error-free performance time of expert users. The models have been validated in a variety of human-computer systems, which has also produced a useful database of empirical performance data. We believe that these models may be useful in design, in the style exhibited in Chapters 4 and 9 and in the example of Section 8.4.

However, as of yet we have only small bits of evidence for the usefulness of the models in actual design situations. Let us cite one interesting application by a product testing group within our own company, Xerox. The group was testing alternative command schemes for a particular set of routine tasks. They taught the schemes to novice users in order to evaluate how easy they were to learn. However, they did not have enough time to train the users to become experts and so could not measure expert performance. Instead, they used the Keystroke-Level Model to calculate the expert performance time. That is, the initial part of the learning curve was measured experimentally, while an asymptote was calculated from the model. Thus, they were able to put together, within their constrained time limits, a fairly complete picture of behavior with the alternative command schemes using both experimental and calculational performance models.

12.3. EXTENSION: AN EVALUATION METHODOLOGY

A study that builds on and extends the work in the present book towards practical application was conducted in our laboratory by Teresa Roberts (for her Ph.D thesis in computer science at Stanford University). The goal of her study was to develop a practical methodology for evaluating computer text-editors. In this section, we briefly describe Roberts's methodology for evaluating text editors, her empirical results, and how these extend the results so far reported (see Roberts, 1979, for the original technical report and Roberts and Moran, 1982, for additional data and analysis).

METHODOLOGY

Roberts began by enumerating a population of 212 text-editing tasks. Each task was expressed in a way neutral with respect to any particular type of text-editor. From this population she selected a set of 32 *core tasks*, which included the basic editing tasks that any text-editor can be expected to perform. This set of core tasks provides a common basis for comparing the performance of different editing systems.

Roberts's evaluation methodology covers four performance variables: functionality, time, learning, and errors. The latter three are measured over the core tasks, whereas functionality measures how well an editor extends beyond the core tasks.

Functionality. In Roberts's methodology, the functionality of an editor is measured by having expert users rate whether each task in her task population can be accomplished with the editor. (The rating levels are: "can't be done," "can be done at manual speed," "can be done clumsily," "can be done efficiently.") Scores are summed up to give each editor an overall functionality rating. The scores can be partitioned into different task categories to show the strengths and weaknesses of the editor.

Learning. Learning is measured experimentally in Roberts's methodology by teaching a novice with *no* computer experience how to do the core tasks with the editor. The experimental learning session is made up of five cycles, each consisting of a teaching part, followed by a quiz to measure what the novice knows how to do (a learning session usually takes from two to five hours). Learning is scored by taking the total time in the session and dividing by the total number of tasks that the quizzes reveal the novice has learned, i.e., the learning time per task. The overall learning score for an editor is the average learning time for the four novices.

Time. The time it takes experts to perform core tasks is also measured experimentally. An expert user of the editor is clocked while performing a benchmark set of about 60 editing tasks (usually taking about 30 minutes). Note that this experiment is similar to our experiment in Chapter 3, except that the times are measured with a stopwatch, rather than with an on-line data collection facility. In addition to the overall time, the time the expert spends correcting large errors (i.e., large enough to be timed with a stopwatch) is also noted. The time score is the error-free time (total time minus the error-correcting time) to edit the benchmark tasks. The overall time score for the editor is the average error-free time for the four expert users.

Errors. Errors are difficult to measure in a simple experiment. Large damaging errors are rare enough so that it takes a long time to collect a reasonable sample. Further, there are large individual differences in error rates, even for routine errors. Roberts explored several methods of assessing errors, none of which seemed satisfactory enough to be used in practice. A modest indication of error effects, however, is the the percentage of time spent correcting errors in the core benchmark experiment above. Thus, the error score for an editor is the average error time, as a percentage of error-free time, for the four expert users.

Cost of Evaluation. An important constraint on this methodology is that it must be relatively easy to use. This is why only manual (stopwatch) measurements are required and why the minimal number of users are measured in the time and learning experiments. The time required to do a complete evaluation of a single editor depends on many factors—the evaluator's familiarity with the methodology, the effort required to prepare the materials for the specific editor, and the difficulty of recruiting users for the experiments. We have found that an editor evaluation takes roughly a week to do for an experienced evaluator.

VALIDATION AND EMPIRICAL RESULTS

Roberts tested her methodology by evaluating four widely-used editors: TECO (BBN, 1973), WYLBUR (Stanford, 1975), NLS (Englebart and English, 1968), and a display-based WANG word processor. We also include here an evaluation of BRAVO, BRAVOX (an extended version of BRAVO), GYPSY (another experimental editor developed at Xerox), and EMACS (Stallman, 1981).

The methodology provides a multidimensional evaluation of the editors. Each editor can be characterized by a 4-tuple of numbers. This summary evaluation is presented in Figure 12.5 for the eight editors, which shows the performance tradeoffs between these editors. The major differences are between the non-display editors (TECO, WYLBUR) and the display editors (all the others). With the exception of NLS on error time and GYPSY on functionality, the display editors are better on all performance dimensions. The display editors are up to twice as fast to use and have about 50% more functionality. On learning, TECO stands out as taking nearly three times as long to learn as the others. One surprising result is the high correlation ($R = .80$) between the time and learning scores. It is usually thought that systems that are highly efficient

Editor	Evaluation Scores			
	Functionality	Learning $M \pm CV$	Time $M \pm CV$	Errors $M \pm CV$
	(% tasks)	(min/task)	(sec/task)	(% time)
TECO	39%	19.5 ± .29	49 ± .17	15% ± .70
WYLBUR	42%	8.2 ± .24	42 ± .15	18% ± .85
EMACS	49%	6.6 ± .22	37 ± .15	6% ± 1.2
NLS	77%	7.7 ± .26	29 ± .15	22% ± .71
BRAVOX	70%	5.4 ± .08	29 ± .29	8% ± 1.0
WANG	50%	6.2 ± .45	26 ± .21	11% ± 1.1
BRAVO	59%	7.3 ± .14	26 ± .32	8% ± .75
GYPSY	37%	4.3 ± .26	19 ± .11	4% ± 2.1

Figure 12.5. Evaluation summary of eight text-editors.
The Functionality score is the percent of the 212 tasks in Roberts's task population that can be accomplished with each editor. The Learning score is the average learning time per task for four novices. The Time score is the average error-free time per task for four expert users on the benchmark set of tasks. (The time scores are large, because many of the tasks on the benchmark required many unit tasks to perform.) The Error score is the average percentage of time the four expert users spent correcting errors; the score is given as a percentage of the error-free time. The *CV*'s show the amount of between-user variance. The evaluation results for TECO, WYLBUR, NLS, and WANG are from Roberts (1979).

to use by experts take longer for novices to learn. This is not the case in this set of editors; the faster editors to use are also faster to learn.

The experimental results of the time dimension were compared against the predictions of the Keystroke-Level Model. The model predicted over 75% of the error-free benchmark time for most of the editors. For TECO, however, the model only predicted 50% of the time. The problem here was that the methods actually used by the expert users were not predicted correctly; the users were much more cautious than predicted in using TECO. When the model's prediction for TECO was adjusted for the actual methods used by the test users, then it accounted for 87% of their error-free time. These predictions are quite reasonable, given the differences between the assumptions of the model and the conditions of Roberts's experiment.

Finally, since these evaluation experiments were run on 32 expert users and on 32 novice learners, they provide us with some useful empirical results on individual differences. On expert performance, there was a factor of 1.5 to 2 between the fastest and slowest users within each editor, which is consistent with the results of Chapter 3. It is interesting and somewhat surprising that there was not a great deal more variation among the novice learners than among the experts, i.e., there was about the same range of ratio between the fastest and slowest learners as between the fastest and slowest experts. By far the greatest individual differences occurred with the error times. Expert users spent from as little as 0% to as much as 28% of their time in errors, averaging 10% error time. Roberts measured error time in real time with a stopwatch, and she had to ignore the small errors. A more careful measurement of errors (on videotape, say, as was done with all the error measurements we have reported) would yield somewhat higher percentages.

APPLICATION

The places where Roberts's study contributes to application have been shown in Figures 12.1 to 12.4. Her study is mainly oriented to the design function of system comparison. Now that her data can be used as a standard of comparison, her methodology also enables the system evaluation of individual editors. The most important aspect of Roberts's work, in the context of this chapter, is that it extends the scope of our studies on two performance variables—functionality and learning.

12.4. ADVICE TO THE DESIGNER

We have presented an approach to applying psychology to design that centers around the notion of performance models. Our implicit advice to the system designer has been to use these models in design. We now present this advice more explicitly in the form of a set of system design principles (listed briefly in Figure 12.6) derived directly from the main results of our studies. Since, as we have seen in Section 12.2, the studies are highly skewed towards certain issues, the principles do not cover the whole spectrum of design concerns. Nor do we attempt to exhaust all the principles implicit in the models; we only present some of the more important and fundamental principles.

1. Early in the system design process, consider the psychology of the user and the design of the user interface.
2. Specify the performance requirements.
3. Specify the user population.
4. Specify the tasks.
5. Specify the methods to do the tasks.
6. Match the method analysis to the level of commitment in the design process.
7. To reduce the performance time of a task by an expert, eliminate operators from the method for doing the task. This can be done at any level of analysis.
8. Design the set of alternative methods for a task so that the rule for selecting each alternative is clear to the user and easy to apply.
9. Design a set of error-recovery methods.
10. Analyze the sensitivity of performance predictions to assumptions.

Figure 12.6. Some principles for user-interface design.

The first few principles summarize some high-level concerns and attitudes about design.

> ***Principle 1:*** *Early in the system design process, consider the psychology of the user and the design of the user interface.*

This may seem too obvious to mention, but it is fundamental and often stated (e.g., Hansen, 1971). If consideration of the human-computer interaction is put off until the computer system is designed, then the psychology of the user will not have any weight among the variety of concerns that face the designer. This principle does not itself tell the designer what to do; the next few principles spell out some concrete actions.

According to Formula 12.1, the human-computer system consists of the task, the user, and the computer, which together determine the system's performance. The designer's job is to specify the total human-computer system. The designer does not have to be told to specify the computer; but he may need to be reminded of the performance requirements, the user, and the task.

Principle 2: *Specify the performance requirements.*

There are many performance variables—functionality, time, errors, learning, etc. Designing to improve performance on one dimension does not necessarily help performance on other dimensions. For example, optimizing the performance time of a system does not improve its learnability (in fact, high concentration on time optimization may make a system harder to learn). There are tradeoffs to be made in performance. For example, using the models we have presented to calculate performance time and using Roberts's methodology for measuring learning, one can quantitatively compare the tradeoffs between ease of learning and speed of execution in a system. Thus, it is important that the designer be clear about his priorities on the performance variables.

Principle 3: *Specify the user population.*

In Chapter 3 we have seen that there is about a factor of three in performance time among expert users—about the same range as the performance among different editing systems. Considering non-expert users, the range of user performance is much greater. Thus, in order to predict the performance of the human-computer system, the designer must know the important characteristics of the user population. If the target population of users is highly varied, it is important to characterize the different kinds of users, for their performances will be quite different. Much of this characterization can be done quantitatively. For example, the Keystroke-Level Model (Chapter 8) shows how the user's typing speed affects his performance time.

Principle 4: *Specify the tasks.*

Performance can only be assessed relative to the set of tasks that must be done. It is not possible to specify all the tasks that the user will want to do. However, specifying a reasonable benchmark sample of tasks is infinitely better than just listing gross task characteristics. The benchmark sample should include representatives of the qualitatively different kinds of tasks the user will face. An example of task generation is given in Chapter 9. The different types of tasks occur with unequal frequency—most of the user's time will be spent doing a very few task types. It is important to specify these high-frequency tasks. The user will become highly skilled on these tasks, and they should be made easy and efficient to do.

Task analysis can be done at different levels of detail, for any task can be decomposed into a task-subtask hierarchy (as was done in Chapter 5). What is the appropriate level of task analysis? In order to keep the range of design possibilities open, tasks should be specified in a way that makes minimal assumptions about the structure of the computer system, except for the structure that is fixed a priori as part of the design requirements. The Unit-Task Level of task analysis, as illustrated in Chapter 9, is the most detailed level of task specification that is practical early in design.

As analysis becomes more and more dependent on system structure, task analysis turns into method analysis. Task analysis reflects more the demands of the external environment, whereas method analysis reflects more the demands of the computer system and the ways in which the user adapts to them. There is, of course, no sharp line between task analysis and method analysis.

Principle 5: Specify the methods to do the tasks.

It is important to grasp the central role that the methods play in determining the level of performance. Skilled human-computer interaction consists of execution of assimilated methods. What makes a user skilled is his highly integrated knowledge of tasks, methods, and the connections between them. System designers tend to concentrate on the commands of the computer system (just look at the documentation for almost any system, which is usually a catalogue of commands). Yet it is *how the commands are used*—the methods—that is most important to the user. Once methods are laid out explicitly, many of the gross aspects of performance can be seen by inspection, even without formal models. For example, particularly long or awkward methods will stand out. Also, it is possible to assess informally the consistency between different methods.[3]

> ***Principle 6:*** *Match the method analysis to the level of commitment in the design process.*

As with tasks, methods can be specified at different levels of detail. In order to predict performance from a method specification, a performance model is required, which in turn determines the method description. There is no single best model; different models are appropriate at different stages of design—depending on the amount of detail known about the system under design.

Several levels of method analysis were introduced in Chapter 5. The Unit-Task Level requires only a modest commitment to the structure of the computer system. This level of analysis is appropriate early in design to assess the task domain by getting a rough picture of the total system performance. It is also useful when trying to decide on major components of the computer system, as was illustrated in Chapter 9. At the Functional Level of analysis, the unit tasks are decomposed into their four functional components—Acquire, Locate, Change, and Verify. This level begins to show how the unit tasks interact, as illustrated in Chapter 9.

The next lower level of analysis is the Argument Level, in which there is commitment to the set of commands and the arguments they take. This level is appropriate while the command set is being designed, but where the small details of the command syntax are ignored. Although we have not given any illustrations, the Argument Level is actually a quite useful level of analysis. For example, this is the level at which the scheme for defaulting arguments can be considered. And finally, at the Keystroke Level there is commitment to the actual keystrokes and other physical operations for executing commands. This level of detail is not appropriate until fairly late in design. But once this level of detail is reached, it is possible to do considerable quantitative analysis of performance time, as we have shown with the Keystroke-Level Model in Chapter 8.

[3] Various kinds of rule-based descriptions of the methods can be used to assess consistency more precisely (e.g., Moran, 1981*a*; Reisner, 1981), although existing rule-system proposals are not yet developed enough to be performance models.

> ***Principle 7:*** *To reduce the performance time of a task by an expert, eliminate operators from the method for doing the task. This can be done at any level of analysis.*[4]

Once methods are laid out, at whatever level, and the performance time calculated, the designer may then want to make the performance more efficient for expert users. All the performance models we have presented suggest that expert performance is composed of a sequence of operators and that the performance time is the sum of time for each of the operators.

Which operators can be eliminated depends on the stage of design and the level of analysis. For example, the performance time for a job can be reduced either by reducing the number of unit tasks or by reducing the time per unit task; however, only the former is possible early in design at the Unit-Task Level of analysis. At the Unit-Task Level, the most likely way to reduce unit tasks is to extend the functionality of the computer system to, in effect, combine unit tasks (e.g., a text-editing function for inserting a pair of parentheses around a piece of text can combine what would otherwise be two unit tasks into one). At the Argument Level, the most obvious way to reduce time is to devise appropriate default values for arguments and even for commands (e.g., with a Redo command). At the Keystroke Level, the way to efficiency is to devise short codes to specify commands and arguments and to eliminate redundant terminators.

> ***Principle 8:*** *Design the set of alternative methods for a task so that the rule for selecting each alternative is clear to the user and easy to apply.*

Alternative methods can be provided to do a task. This allows the different methods to better take advantage of the specific structural features of the different task instances. From our study of method selection in Chapter 5, we characterized the expert user as having simple decision rules for selecting an appropriate method in each task instance.

[4] This principle might well be called the "Gilbreth Principle," for Gilbreth (1911) was one of the first to systematically code behavior into a sequence of physical movements (which he called "therbligs," but which we would call "operators") and to optimize performance by eliminating unnecessary movements. Gilbreth, however, did not have any notion of levels of analysis; all his analyses were at the same level of physical movement.

Another useful notion from the method selection study is the notion of the "default method," the method that the user selects by default in preference to other alternatives. The default method is not the most efficient method, but it is a general method; other alternative methods are more specialized, but more efficient. Categorizing methods this way provides a good strategy for designing method alternatives: provide a general-purpose method plus a set of efficient special-purpose methods. A similar strategy is to design alternative methods for specifying commands, easy-to-remember but slow methods (such as typing out the command names) and fast but harder-to-remember methods (such as special single-key codes). These strategies allow incremental learning. The novice user need only learn the general-purpose, easy-to-remember methods at first; he can acquire the more efficient methods one by one as he becomes more expert.

Principle 9: *Design a set of error-recovery methods.*

Another aspect of expert performance is errors. We have seen that expert users adopt strategies that permit up to about 30% of their time to be spent correcting errors. Error-correction is also highly skilled behavior. Thus, error-recovery methods should also be designed for learnability and efficiency. This suggests, for example, that an Undo command would be worthwhile. In designing the Undo command, consider carefully how it will be used to help the user recover from specific kinds of errors, for all errors are not the same in the scope or severity of their effects.

In the analysis of errors, it is useful to separate the occurrence of errors from the treatment of errors once they occur. We do not yet have any models to help predict errors, although common sense can suggest a few sources of errors, such as the user accidentally hitting an adjacent key on the keyboard.[5] But given that an error has occurred, the expert user's handling of the error is a skilled activity and is thus amenable to quantitative analysis by the performance models of the sort we have described.

[5] Such an accident would be a motor "slip." See Norman (1981) for a categorization of action slips, including cognitive slips. For some evidence that error occurrences can be modeled, at least in closed task domains, see Brown and VanLehn's (1980) Repair Theory model of the sources of "bugs" in arithmetic procedures.

Principle 10: *Analyze the sensitivity of performance predictions to assumptions.*

In carrying out any kind of performance analysis, the designer must make assumptions about the user (psychology does not have all the answers), about the computer system (the design is not fully specified until the end), and about the task environment (which cannot be fully anticipated). Thus, any predictions of human-computer performance should be checked for their sensitivity to these assumptions.

One of the main advantages of symbolic models, as we have emphasized, is that they allow unknowns to be parameterized and hence to be analyzed for their effects on performance. This use of parametric and sensitivity analysis was illustrated in the examples in Chapter 4 and in Section 8.4. Although performance usually does change, it often does not change in ways that affect the design decisions that motivated the performance analysis. Even if sensitivities are found, it is much better to make design decisions knowing what factors critically effect the decision and what factors do not matter. With such knowledge, it is possible to know which previous design decisions must be re-evaluated as new knowledge develops during the design process.

12.5. CONCLUSIONS

We have proposed that an applied psychology should take a particular form in order to be of use in the design of interactive human-computer systems. The central feature of this applied psychology is the packaging of psychological knowledge into performance models that can predict the performance of the human-computer system from specifications of its structure. The design process can be decomposed into several different kinds of design functions, most of which require the use of performance models.

In this book we have concentrated most heavily on performance models for calculating expert performance time.

Roberts (1979) has extended the use of our models by developing a practical methodology for evaluating text-editing systems along four performance dimensions—functionality, learning, time, and errors.

We have expressed some of the results in this book as a set of design principles to aid in the design of systems for human-computer interaction.