INTERFACE DESIGN ISSUES FOR ADVICE-GIVING EXPERT SYSTEMS

Advice giving could become the first successful domain for intelligent interfaces.

JOHN M. CARROLL and JEAN McKENDREE

Two empirical phenomena have structured much recent research on human–computer interaction; these are (1) that people have considerable trouble learning to use computer systems (e.g., [60, 63]), and (2) that their skill tends to asymptote at a relatively mediocre level (e.g., [69, 74, 78]). There would appear to be a conspiracy of reasonable strategies for learning and performance in a practical domain that has the surprising effect of undermining the motivation required to learn and to perfect skills [21].

The situation can be sketched as follows: People want to use computer equipment because they want to get something accomplished. This is good in that it gives users a focus for their activity with a system and increases their likelihood of receiving concrete reinforcement from their work. But this same pragmatism can also make an individual unwilling to spend any time learning about a system on its own terms. After all, to consult on-line tutorials or programmed self-instruction manuals is for a time to effectively cease working. There is then a conflict between learning and working that inclines new users to try to skip training altogether, or to skip around in a training sequence, sometimes with disastrous consequences. It also inclines more experienced users to stagnate, for when situations occur that could be more effectively handled by new procedures, these users are likely to stick with the procedures they already know, regardless of their efficacy.

Recently, many researchers have rallied to the suggestion that the motivational "cost" of learning and skill improvement could be reduced through the use of intelligent system monitors (see, e.g., [50]) that could mitigate the learning versus working conflict by better integrating the time and effort spent on learning with actual use of a system. This "advice-giving" approach contrasts with the more typical drill and practice style of contemporary on-line tutorials and the confusing verbosity of typical context-insensitive help commands [49]. On-line tutorials and helps do in some cases monitor and adjust to the needs of specific users, but characteristically maintain a sharp separation between learning and working that can undermine the user's motivation to make use of training and help materials.

We present a structured overview of current proposals for advice-giving systems in terms of three specific goals. First, we review the literature. Research in this area is expanding rapidly, and the time has arrived when "intelligent" monitoring technology is practical. This makes it important to structure and assess the research literature now.

Second, our particular orientation is to consider requirements for such systems from a *behavioral* standpoint. One striking lesson in recent computer science is that the capability to build a new system technology is only one condition for the potential "real success" of the technology. Mere technological feasibility must be augmented by empirical study of whether and how people will find a new technology useful and tractable. It is pointless to build such facilities unless we take into account behavioral requirements on their usefulness and usability.

Ideally, such an assessment can be made even before the new technology is completely developed (e.g., [45, 61, 80]).

Our third goal is to extract the themes and conclusions in this research that can help to direct future work. To lay our cards on the table from the outset: We believe, based on our study of the current literature, that far too little behavioral work has been invested to date in design research on advice-giving systems. This is ironic since one of the chief practical motivations for such systems is to produce advice-giving capabilities that effectively support the needs of human users. And, indeed, this lack of attention to usability is increasingly being identified as a key reason for the limited impact that expert-system technology has had on real computing (e.g., [27, 54]).

We have organized our discussion of the literature into two major topic areas: the knowledge involved in giving advice, and the contingencies for and content of effective advice.

KNOWLEDGE REQUIREMENTS

Advice-giving systems will need to have knowledge if their advice is to be worth anything. What will it mean for a system to have knowledge? What does an advice-giving system need to know about? How much does it need to know? How is it supposed to get this knowledge? Philosophers can rest easy; for today's computers, to "have knowledge" means no more than to be able to store information and to act on the basis of that information. Advice-giving systems are distinguished chiefly in that they store information about a system and its commands, conditions, procedures, etc., and can access this information and provide it to users as on-line training and help. This puts the philosophical issues to rest, but it awakens a host of technical issues.

In the editing environment in which this article was composed, the command "help split" evokes six screens of information defining the command *split*. The system has stored knowledge about itself and can provide this knowledge on request. True advice-giving systems will have to know about more than merely their own dialogue conventions. They will have to know about *advice-giving*, about *the tasks they are to be used for*, and about *the ways that users can vary*. For example, our editor might be enhanced to know to respond differently to "help split" depending on whether a text file or a Pascal program is being edited [9, 49].

We consider these types of knowledge under three headings. *General skills* includes knowledge about things like tutoring styles and natural language—knowledge areas that transcend the particular system, the user, and the tasks for which the user employs the system. *Domain knowledge* (as in the example of elaborating "help split") incorporates specific task information. A *user model* is what the system knows about the person interacting with it.

General Skills

The importance of general skills is often acknowledged, but rarely discussed. General skills encompass knowledge of such vast and intricate topics as advisory strategies and natural language. Our understanding of these areas is very incomplete, and it is not surprising that we have been unable to satisfactorily codify these knowledge domains in advice-giving systems.

Advisory Strategies. How can advice be given effectively? How can a particular advisory strategy be modified effectively on the basis of feedback from an advisee? Sleeman and Brown [87], in the introduction to their well-known anthology on intelligent tutoring systems, label the need for explicitly describing different advisory strategies as "critical," but the papers in their book do not in fact address the issue. Clancy [24], for example, takes it as a significant advance that his GUIDON system architecturally separates the domain knowledge base from the advisory strategies—underscoring the importance of the latter. He [24, p. 205] specifically calls attention to the question of evaluating the strategies, but then leaves the question open.

Typically, researchers have opted for a particular advisory style, offering little or no empirical rationale. For example, many tutoring systems employ a Socratic style where the system poses questions and the user is expected to provide answers [24, p. 219; 88]. It is likely that this is an effective style for interactive tutoring in many situations, but it is also significant that no evidence has been offered to support this assumption. The possibility exists that the Socratic style is often adopted for tutorially irrelevant reasons. Giving the system control of the dialogue by allowing it to Socratically pose all the questions allows a simple question-list knowledge structure (e.g., [55]). The trade-off is that system control of advisory dialogue may make the system behaviorally unacceptable [14].

Coombs and Alty [26] analyzed actual interactions of computer consultants working on problems that users presented. They found that the flow of information tended to be one way: After the user posed an initial question, the advisor assumed control of the interaction. Advisors rarely included explanations or checked whether the user understood the advice. Users often criticized advisors for "not mak-

ing the information meaningful" [26, p. 407]. Advisory interactions that were judged successful by users tended to be ones in which the advisory strategy allowed shared control of the dialogue (see [27]).

Learning-by-doing environments are the other major advisory design that has been implemented experimentally [10, 12]. In this approach the user can more freely initiate actions. Each user move is compared with an expert move, as generated by the system, and feedback is provided to shape the user's responses toward the expert prototype. This approach has the advantage of placing more initiative, and therefore more control, in the hands of the user. However, the increased flexibility in user interaction may entail more demanding knowledge requirements for the system, which must know every way a user's actions can depart from those of the normative expert, and the particular significance of each potential departure.

Most learning-by-doing environments have been developed for educational games where the state-space of possible move combinations is relatively small. Several investigators have developed empirically based taxonomies of "bugs" [10–12] or of "object-related misconceptions" [64]. But these taxonomies are very domain dependent and hence may not capture general knowledge about strategies for error correction or the development of expertise by means of advice giving (cf. [71]).

Neither the Socratic nor the learning-by-doing approach enjoys much systematic empirical validation (but cf. [11]), and a researcher's choice of advisory strategy remains largely a matter of fiat. Kimball [55, p. 283], for example, explicitly rejects learning-by-doing environments for the domain of posed math problems, but offers no empirical rationale whatsoever. It is also notable that both the Socratic and learning-by-doing approaches are always applied unilaterally in current tutoring systems. Current systems are not able to reason about tutoring per se or to select situationally appropriate tutoring strategies dynamically.

Recently, Fischer, Lemke, and Schwab [36] acknowledged the need for a variety of advice-giving strategies. They distinguished between active and passive strategies for help (in active strategies the system interrupts the dialogue to provide advisory commentary on the user's actions; in passive strategies the user must explicitly request advice). Unfortunately, they did not show empirically how either of their advisory strategies was specifically called for on behavioral grounds, or how implementing them in fact proved useful.

Natural Language. As with advisory strategies, there has been a general recognition of the importance of natural language to successful advisory interfaces (e.g., [38]). Unfortunately, the problem has often been finessed rather than confronted. Systems like those described by Coombs and Alty [27] and by Goldstein [41], for example, appear to have little or no natural-language capability. Fischer, Lemke, and Schwab's Passivist system [36] merely slices out keywords in the style of Weizenbaum's Eliza [95]. Sleeman's ACE system [86] parses with simple argument templates, a very rough and inevitably fragile and domain-specific approach. What are the behavioral consequences when the user inadvertently foils the template and gets garbage advice from the system? The risk, as Brown, Burton, and deKleer [10, p. 244] observe, is that arbitrarily approximate treatments of natural language may cause more problems than they solve.

Malhotra and Sheridan ([61]; see also [90]) adopted an empirical approach to this problem by simulating an order-writing and invoicing system that could answer questions and respond to commands in English. In their studies, the user–system dialogue was filtered by the experimenter, who could intervene behind the scenes to enhance the apparent natural-language capabilities of the system. In this way, Malhotra and Sheridan were able to study the behavioral requirements for natural-language capabilities in a system without actually implementing them. Over half of the sentences produced by their subjects instantiated a small set of structural templates that they were able to describe. However, more than a third of the sentences produced were classified as not syntactically analyzable.

Chin [23] has more recently employed this empirical approach to study a simulation of the UNIX® Consultant [96]. He found that over a quarter of the English queries submitted to the simulated consultant incorporated contextual syntactic constructs, such as ellipsis, anaphora, indirect speech acts, and grammatically incomplete clauses. However, the incidence of contextual constructs in a control group querying a human consultant was nearly twice as great, suggesting on the one hand that natural-language interface facilities need to be able to interpret contextual constructs, and on the other that people may voluntarily restrict the contextual complexity of their queries when interacting with an advisory system.

Both of these findings are encouraging for the possibility of developing empirically adequate but computationally limited natural-language capabilities. They provide an alternative to natural-language restrictions motivated chiefly by ease of implementation. Of course, they also raise the theoretical issue

UNIX is a trademark of AT&T Bell Laboratories.

of how to more comprehensively characterize the nature of appropriate natural-language restrictions. Finally, they begin to provide methods for analyzing the cost–benefit trade-offs in general skill requirements for advice-giving systems (e.g., how much does a particular enhancement actually enhance the usability of an advisory interface?), which is a critical issue if these systems are to be practical in the near future.

General skill areas are clearly amenable to behavioral study. We hope that simulation techniques like those of Malhotra and Sheridan [61] and Chin [23], and direct behavioral investigations like that of Coombs and Alty [26, 27], will be more widely employed in future explorations of general knowledge requirements for advisory expert systems.

Domain Knowledge

In our "help split" example, we posited an on-line help system that could refer to heuristic knowledge about the tasks it supported in determining how to respond to help requests. The system described had a domain knowledge capability of extremely limited scope. Many diagnoses that an actual advisory system might make about a user's intentions would require examining and extracting a pattern from a sequence of user activities (e.g., [12, p. 96]) and looking not merely at the state the user has arrived in, but at the entire process that the user traversed in getting there [40].

Psychological Pertinence. This issue is more than merely one of *how much* user activity is taken into consideration. It has become a question of interpreting the user's actions in terms of the user's intentions. Fitter [37, p. 341], for example, argues that in order to be useful to people a system must represent its knowledge *as people know it* (see also, [38, 51]); that is (to quote Fitter),

> (1) the underlying process which the computer is performing should model the processes which are directly pertinent to the user in a manner compatible with the user's own model of the process; and
> (2) the communication language (or user interface) should be designed so as to reveal [a system's] underlying processes as vividly as possible.

The rub, of course, is to accurately carry out these prescriptions in given cases (and to be able to know that you have done so). What does it mean to model a system's operation and interface on processes pertinent to the user? What, after all, is the user's internalized mental model of the system?

These are difficult questions. The research literature in psychology provides little basis for deriving serious mental models of complex task domains [16], and virtually no direct behavioral research either informs claims about psychological pertinence or has been brought to bear on evaluating these claims. It is worrisome, then, to find that some researchers merely assume that these questions are settled. Jagodzinski [51, p. 218] states that "since the inception of computers there has been a continuous movement in programming away from methods which reflect the working of the computer towards methods which correspond more closely with human cognitive processes." In a very rough, intuitive sense, this claim seems correct. But, as a summary of the state of research, it is a meaningless promise of optimistic intentions, lacking any serious probing of what "human cognitive processes" might refer to.

A more fruitful approach might be to focus on the apparent discrepancies between human cognition and computational knowledge techniques. Kidd and Cooper [54] called attention to the mismatch between the exhaustive backward chaining strategy for drawing out connections in a database and what they call the "quick stab" strategy, perhaps more typical of how people try to draw out such connections. They did not analyze this discrepancy in much detail or study its empirical consequences directly, but their suggestion takes a step toward identifying relevant human cognitive processes.

Multiple Representations. Fikes [35] and Stevens, Collins, and Goldin [88] raised the important possibility that multiple representations of a domain are necessary. The idea is that any single domain representation will not provide enough basis and flexibility to support effective advising. Fikes contrasted rote description, functional description, and procedure teleology description. Rote description focuses only on the steps of a procedure and the desired result. Functional description includes also a rationale for each step—its dependency relations with other steps and its contribution to the desired result. Procedure teleology extends this rationale to the procedure itself by describing the task that the procedure is designed to achieve and the function that each step in the procedure plays. A limitation of Fikes's work is that it proceeds from a purely a priori standpoint: The specific empirical consequences of his notion of procedure teleology remain unclear.

Stevens, Collins, and Goldin [88] developed their multiple representations thesis more concretely—in the context of an empirical investigation. People were taught facts about weather (e.g., Oregon is wet) and the mechanisms that underlie these facts. They identified two relevant levels of knowledge: scriptal knowledge, expressing situationally grounded, causal relations; and function knowledge, expressing

a more abstract, qualitative model. They found that scriptal knowledge governed the sequencing of major topics in an advisory dialogue, whereas functional knowledge governed the finer structure of advisory interaction. Unfortunately, their work rests on an ad hoc sample of pedagogical anecdotes and presents little systematic evidence that their distinction actually underlies a significant enough distinction in human thought to constrain the design of advice-giving systems. Nevertheless, this work is empirical and suggests an interesting direction for further work on knowledge representation.

Toward a Theory of Domains. Lurking around all work on domain knowledge is the disturbing possibility that the research outcomes will be too strictly domain specific. Many of the domains that have been investigated so far are extremely simple. VanLehn [93, p. 9] described his own domain of interest (subtraction) as "dry, formal and disconnected from everyday interests." This raises a general question about the adequacy of the solutions proposed: Will they extend to more complex and typical arenas of knowledge? It is also notable that much extant work has focused on what one might call declarative, as opposed to procedural, domains. Procedural domains include the interactive use of systems, program composition, and debugging. The meteorological domain studied in [88], for example, includes underlying mechanisms of cause and effect, but no procedures. This raises the question of how the analysis of multiple knowledge representations might extend to a procedural domain: for example, what is the scriptal/functional distinction in such a domain?

What kinds of domains are there? Naturally enough, computer science application work has focused attention on procedural domains. At an extreme from these are purely factual domains (foreign language vocabulary, history). Somewhere in between, perhaps, are functional domains (mechanics, meteorology). What kind of domain is music, though? We need to better articulate a taxonomy of domains that expresses the similarities and contrasts between types of domains. This taxonomy must reflect the significant empirical contrasts between types of domains, and so clarify what sorts of domain knowledge theories we have and do not have at present [39, 82, 92].

The Grain of Analysis. It is often claimed that representing more domain knowledge is a key to better advice-giving systems. Stevens, Collins, and Goldin [88, p. 22], who represented their knowledge domain in two separate formats, suggested that additional multiple representations are necessary. In a similar vein, VanLehn [93, p. 23] concluded that his own progress would have been greater if he had represented domain knowledge at a finer grain—the processes underlying procedural bugs rather than the bugs themselves. Indeed, to predict that more could be achieved if only more knowledge and finer grain knowledge were represented is almost a given in the knowledge-based systems literature.

Rich [76] proposed that a convenient grain of representation for domain knowledge is the level of statements in the system code. She conjectured that if an advisory facility could examine system code, statement by statement, it could find answers to what she took to be the principal types of user queries ("What causes this outcome?" "What will happen if I do this?" and "What is the difference between these two ways of doing it?"). This is an interesting proposal if only because it gives up on the requirement of psychological pertinence that most researchers have placed on their views of domain knowledge. However, it is important to bear in mind that Rich did not demonstrate that her proposal could really produce a feasible domain representation, with respect to answering the user query types she enumerated. Moreover, she did not establish that these query types are in fact exhaustive, or even typical of actual user queries.

Reiser, Anderson, and Farrell [75] gave a user-based rationale for selecting Lisp atoms as the grain of analysis for an intelligent tutoring system for Lisp. They argued that a finer grain (e.g., keystrokes) would often underdetermine specific errors and thus underdetermine specific tutorial advice. A larger grain, however, would delay feedback past the point when specific errors could have been classified, which allows the user to continue on past the error. By focusing on a single and salient grain of resolution for domain knowledge representation (like the Lisp atom), the system might present a more consistent advisory style to the user. However, selecting the atom as the grain may be a double-edged sword, since providing advice at the atom level might cause the system to overlook or misadvise on problems that in fact pertain to higher levels of program structure or involve conceptual tangles of several lower level problems.

Instead of stuffing more domain knowledge into advisory systems, we might ask, How little domain knowledge, and at how large a grain, is optimum for advice giving? This would focus relatively more attention on projecting dialogue fluency and the appearance of some advisory competency and relatively less on the organization and access of very large knowledge databases. Thus, instead of storing the component processes underlying 500 bugs, we might store a dozen prototype scenarios to roughly classify user situations and advise at that grain. Clas-

sic demonstrations like Weizenbaum's Eliza [95] suggest the power that mere appearances can have for people. Our work on scenario machines (to be reviewed later) has also explored this direction.

User Models

Referring once again to our "help split" example, we might consider enhancing the system by maintaining a record of a given user's work patterns and help calls for use in diagnosing the nature of a future help call by that user. A special concern here is that the computer not be, or be seen as, spying on the user (e.g., to benchmark worker productivity). However, to provide individually and situationally pertinent advice, the system may need to analyze user activity. The development and maintenance of predictive user models has been seen as critical for advisory systems (e.g., [4, 87]).

Normative Models. A simple approach is to develop and refer to a single, normative user model. An example of this approach is the training wheels interface studied by Carroll and Carrithers [18], which hard coded the assumption that new users of a word-processing application would not need to access specific advanced system functions. When users attempt to access these functions, they are informed that the functions are not available to them.

The training wheels design was based on empirical research indicating that new users often access an advanced function by mistake, and then become distracted and confused by the consequences. A limitation with this approach is that users could become frustrated with the implicit and inflexible guidance should their actual goals fail to accord with the normative model. Reiser, Anderson, and Farrell [75] employed a normative model with prescribed goals in their Lisp tutor, but then adjusted the advice dynamically in response to specific user actions. Error-correction advice was adjusted to suit the manner in which an error was committed; users could also request specific clarifications or explanations. A limitation of such approaches is that there can be psychological distinctions to a user when there are no functional distinctions to a system [85].

An ambitious approach to advice adjustment is to incorporate both a normative reference model and a user-specific model of a given individual. Sleeman and Brown [87, p. 5] discuss using a comparison of performance outputs of the two types of models as a basis for generating advice. Goldstein [41, p. 67] criticized this approach for assuming that the knowledge of the novice could be viewed as a simple subset of the knowledge of the expert, and that the transition from novice to expert consisted of nothing more than the accretion of knowledge, gradually and ballistically minimizing the performance differential between the two models. Goldstein's discussion echoes critiques of the accretion learning theories of stimulus–response psychology (e.g., [91]) by contemporary theorists in artificial intelligence [72] and cognitive learning [56] who believe that learning involves radical cognitive restructuring.

The possibility that the relation between novice and expert skill is more intricate and perhaps more erratic than a simple quantitative comparison implies is consistent with an empirical test of the ideal normative model approach. Sleeman [86] employed an ideal expert model in designing a system to coach algebra. The model made relevant diagnoses for less than half of the errors that actually occurred in an algebra tutoring situation and made the correct diagnosis on only half of those. The Lisp tutor of Reiser, Anderson, and Farrell [75] introduced a variation on ideal normative models in which the "ideal" model was not of an ideal expert, but rather of an ideal learner—an advanced student [5]. The tutor stored a list of productions as an ideal student model, and annotated this list on the basis of the given individual's monitored performance to indicate which productions were assumed to have been learned. This system correctly diagnosed 45–80 percent of user errors.

Vocabulary Analysis. We have not yet addressed the question of how a user model is to be constructed and updated. Can the system merely ask the user to indicate a skill level? We doubt the value of such an approach. A simple partitioning into "novices" and "experts" is probably not adequate [30], and offering more categories might make the self-classification task difficult and unreliable. People are notoriously bad at giving accurate descriptions of their own information needs [70, 77]. The obvious alternative is to infer the user's skill level (and the user's specific problems and concerns) from actions and responses. This would provide more reliable classifications, but would also make greater demands on the system.

Rich [77] describes an advice-giving facility for a document formatting system in which a user's skill level is determined by a keyword analysis of the vocabulary employed in help calls. Each text formatting command is associated with a hierarchical structure of explanations ranging from general to technical. The skill level diagnosed in the user's help call determines the vocabulary level of the explanation produced by the system. One problem with this approach is that it is not clear how diagnostic the vocabulary in a help call really is: An utter novice will probably not pose very technical queries, but a user with considerable experience might use relatively general vocabulary and there-

fore get advice at too elementary a level. No behavioral assessment of the vocabulary analysis approach has been made yet.

Behavior Analysis. Perhaps a more direct route to automatically diagnosing user skill is to monitor and evaluate the user's actual behavior. Much of this work has focused on monitoring and evaluating user errors. Coombs and Alty [27] stored prototypical error patterns as a normative user model for users of a Prolog environment. When particular users generated patterns of behavior that matched a stored prototype, relevant corrective advice was produced. We would of course prefer to have more than a mere list of attested error types to direct a behavior analysis. We would like to have a more general, or abstract, taxonomy of error, and more importantly, a theoretical understanding of why errors occur.

We have referred to work on the concept of knowledge "bugs," or systematic departures of a user model from an ideal model [10–12]. A bug is defined to be the smallest change in a correct procedure that would make it into an error [11]. Bug theorists stress that this level of analysis is more abstract than that of prototypical error patterns, where errors are simply flaws in the stream of behavior. Bug theorists see bugs as knowledge hypotheses: Errors, characteristic nonoptimalities, and even correct performances could all be evidence of a particular bug hypothesis. Systems like BUGGY and DEBUGGY try to recognize bugs and compositions of bugs in a person's work. Recognized bugs comprise a level of analysis of the user's skill, and a basis for generating specific advice.

Unfortunately, research on behavior analysis is still quite inconclusive. More than five years worth of development work on bug diagnoses for the domain of simple arithmetic managed to analyze only a third of actual learner errors. Perhaps more worrisome is the fact that no principles have been isolated that distinguish learner errors that are not based on misanalysis from those that are. There is still no systematic theory of bugs or of error prototypes. An inventory of bugs or errors is an inventory of mini-theories of a person's knowledge, but there is no systemic constraint on this taxonomy. Conversely, a view of learning as a process of debugging "least variants" in performance is still no richer than the simple accretion of stimulus–response connections. This is a practical as well as a theoretical limitation. Since a particular inventory of bugs or error prototypes develops on a case-by-case basis without any overarching framework, each new case is unique. From a practical standpoint this is a crippling limitation.

Perhaps current approaches to knowledge bugs are at too low a level to capture cognitive skill. One could drop the requirement that bugs differentiate "least variants" in performance and consider only "higher level" bugs in the hope that they might comprise a more efficacious and more tractable grain of theoretical analysis. (Indeed, higher level bugs might coextend better with the level of error prototypes and thus be a more empirically straightforward level as well.) Finally, it is clear that in many cases analyses of user error must explicitly incorporate the intention the user had in making the error [52]. Thus behavior analysis may need to move in the direction of cognitive analysis [94].

Individual Differences. Coming at these problems from the top down, instead of from the bottom up, we might begin with a general analysis of individual differences and so deduce patterns of errors or bugs that are diagnostic. This strategy holds out the promise of providing an integrating framework that could help to generalize the case-by-case behavior taxonomies across task domains and systems.

The user modeling work that underlies the design of advice-giving systems has typically presumed that users are homogeneous in relevant ways, although hardly anyone would dispute that a better alternative would be to tailor interface presentation for individual differences (e.g., [77]). We agree that individual differences may figure importantly into future advice-giving technology, but we also believe that the area is virtually undeveloped at present. It makes sense to say that people have diverse styles and needs for information. And it seems that people who provide information to others (e.g., reference librarians, people giving directions) do make adjustments based on some reckoning of individual difference [57]. Therefore, it does seem reasonable to predict that computer systems may need to incorporate such capacities. What seems unclear at present is just what the relevant dimensions of this accommodation are.

Many of the examples that have been discussed are quite unconvincing. On the one hand, many of the individual difference categories that are discussed are extremely general (like gender and Rich's stereotypes like "intellectual feminist"). The problem with these categories is relating them in focused ways to significant aspects of human information processing: Do intellectual feminists have different advisory needs?

On the other hand, more specific individual difference categories often provide few real design implications. In making a case for considering individual differences in interface design, Rich [77, p. 200] calls attention to studies by Card, Moran, and Newell [13] indicating that keystrokes per task should be mini-

mized in designing word processing interactions for experts, and to studies by Ledgard, Whiteside, Singer, and Seymour [58] indicating that English-like, full-word commands should be used in designing word processing interactions for novices. She calls these "conflicting requirements," but it is not at all clear that they do conflict—even for a specific level of user experience (e.g., Card et al. clearly view keystroke estimates of complexity as approximations, and Ledgard et al. in fact used very *brief* full-word commands).

An interesting and increasingly relevant individual difference is that users' experience with other systems creates specific expectations about functions, vocabulary, predicted errors, and confusions. Rosson [79] found that the number of different full-screen text editors a person had previously used predicted other user characteristics (e.g., the range of program function keys used in editing). Again, individual differences seem ripe for empirical investigation although little has been done as yet.

Knowledge Bounds

How much knowledge is enough? Clearly, it is sound to argue that more explicit knowledge about advisory strategies, more natural-language capability, more content knowledge about given domains, and more extensive user modeling of individual users could make interactive advisory systems more effective. However, this detaches the issues from the problem context that gave them meaning. It is the constraints and limitations placed on knowledge representations that give this work scientific interest. Moreover, developing advice-giving systems is more than an exercise in knowledge representation: The argument for more of every type of knowledge fails to take account of what engineering has to offer.

Knowledge requirements work needs to move toward placing constraints and limitations on knowledge representations, and then measuring and evaluating the ensuing behavioral consequences. In the theoretical arena, this means going beyond inventories of knowledge elements to theories of knowledge structures; it means developing a theory of domains, and better theories of particular domains—empirically testable theories; it means understanding advisory strategies and restricted natural-language capabilities so that alternatives can be explicitly characterized, contrasted, and evaluated, not merely selected or rejected on intuitive grounds.

In the practical arena, we need to look not merely at how to implement more comprehensive approaches, but also at the consequences of self-limiting approaches that are more technically feasible and that may be good bargains in advisory effectiveness. At the Watson Research Center, we have explored a research and training tool called a "scenario machine" [19, 67], which statically encodes a series of goals and tasks for the user to accomplish with the system. The system makes each succeeding task seem plausible as the user works through the programmed scenario via incidentally presented information (e.g., the user receives electronic mail containing a query about a bulletin-board item that must in turn be accessed and examined to answer the query). The scenario machine can provide advisory dialogue about the current goals, the steps of a procedure being executed, or other information relevant to the current task. The scenario that the user traverses is designed to include the fundamental functions of the system—those functions users will need to have understood when they are engaged in actual use of the system. This sort of approach allows the system to provide very appropriate and context-sensitive advice without explicitly representing knowledge about the domain or dynamically inferring the user's goals.

Advisory Knowledge Issues

What advisory strategies are there?
Under what conditions are different strategies effective?
How can different strategies be integrated?
In what specific ways can natural language enhance system usability?
In what ways do people voluntarily restrict their use of natural language when interacting with a recognition facility?
How can a user's mental model of a task domain be incorporated into an advice-giving system?
What properties of domain models, and at what grain of analysis, are most important for generating advice?
How can we generalize advisory techniques across different task domains?
Can user models that incorporate learning transitions and trajectories (as well as end states) be developed?
Are there application areas for which normative modeling approaches will be adequate?
How should individual user models be incorporated into advice-giving systems?
What behaviors and self-descriptions can be collected to build user models?
What differences in individual users are important for advice, and how can they be addressed through design?

A topic like "knowledge requirements" might seem like a "formal" issue—more artificial intelligence than human factors. We think otherwise. In fact, looking over the current literature, it is striking that many of the questions we have raised (and for the most part had to leave open) could *only* be resolved by making behavioral measurements of people using current research prototypes.

ADVICE CONTINGENCIES AND CONTENT

Knowledge requirements are a foundation but at the same time merely a preliminary to the "real" problem of giving advice. Granting that the system knows the right sorts of things and has all the right answers and skills, how should these be structured and deployed in an advice-giving dialogue? What should the system say, and when?

Initiative

Our "help split" example assumes a user who explicitly requests advice by issuing a command. This is quite typical in the commercial state of the art, but we can imagine a system that could directly monitor the user's activities, keeping a log of every user action, analyzing this log, and then initiating advisory dialogue. Indeed, in speaking of advice-giving systems, the implication is that initiative is at least shared in part with the system.

Certain questions follow from this line of reasoning: What level of user activity should be monitored?—that is, what classes of events should be logged? The system could log every single keystroke event, or it could log events at a higher level (for example, exemplars of a closed class of taxonomized errors). How should advice be delivered? The system could merely suggest more efficient or more correct ways to use the system, through verbal feedback, for example, or it could compel the user to do things in these alternate ways. These contrasting approaches rather severely modulate what it means to give "advice."

System Monitoring. In current advice-giving systems, the most typical class of user actions to monitor for is errors. This makes good intuitive sense. After all, it is in the context of an error that users need most to be advised, and it seems reasonable to assume that in attempting to recover from an error a user is motivated to attend to and to use advice. As such, error monitoring provides good grounds for feasibility demonstrations.

The key problem in error monitoring is defining what is to be taken as an error. There are simple cases: In a factual domain, if the user asserts false facts then he or she has made an error and can be corrected and tutored. Even in procedural domains there are fairly simple cases: The user who selects "Printing" before having created any printable data is probably making an error. The user who selects "Application Customizing" at the first log on and prior to any other selection is probably making an error and could be directed to first try out some simpler function. The user who queues and requeues the same print job over and over without ever operating the printer is probably making an error and could be coached on using the printer.

A user action is often an error only with respect to specific user goals. For example, the user who queued and requeued the same print job without printing was making an error only under the assumption that his or her operative goal was in fact to print the job out. If the goal was to fill up the queue, the entire action sequence might have been not merely correct but optimized. Thus, the problem in error monitoring becomes one of diagnosing errors based on inferred goals [52]. It can be very difficult for a system to correctly infer user actions and user goals in an interactive procedural domain like programming or word processing. An even greater difficulty is the fact that errors tangle in sequences of user behavior. A typically correct user action executed in the context of a prior error may need to be interpreted as a consequent error. For example, correctly operating the printer could be an error if the goal is to print Document A and, via some other error, Document B was queued for printing before Document A was queued. It is an error because in this context correctly operating the printer will cause Document B to print.

Error monitoring is facilitated if the user's actions are predictable or directly prescribed. Indeed, this is why so much work on advisory systems has focused on intelligent tutorials rather than on intelligent help systems. Tutorials provide the system with a more restricted domain of action to diagnose. Nevertheless, this also makes clear that work on advisory systems will need to develop toward monitoring both for user problems other than overt errors (e.g., for problematic or merely inefficient patterns of use [78]) and for errors in real contexts of use, outside of tutorial environments. Shrager and Finin [83] describe an advisory system that monitored actual use and suggested more optimal methods to users.

User Discovery. Granting that the system can successfully monitor for, say, errors (relative to appropriately diagnosed user goals), a further question is whether even this is an adequate basis for generating advice. System-initiated advice, however relevant to the user's situation, could be a distraction to a user whose current goal is something other than attending to advice. Fischer et al. [36] acknowledged

this problem and employed a simple time-out approach with advisory interruptions occurring with an arbitrarily bounded maximum frequency. But merely controlling the rate of interruption may miss the point: The system needs to distinguish opportunities for welcomed interruption from advisory harassment.

Proponents of "discovery learning" often advocate waiting until the user explicitly asks for help before offering feedback [34]. The practical problem with current user-initiated advice facilities is that they often require lengthy and tedious prompting dialogue interaction [54]. There is a need for work on schemes for streamlining system response protocols for user-initiated advice requests.

Taking an even stronger discovery learning approach, we might consider downside aspects of providing advice at all. It is often argued that providing information to people is in many cases less effective than allowing them to discover on their own initiative [84]. The discovery approach takes advantage of opportunistic learning, that is, making the most of each unique personal experience. On the other hand, allowing the user to wander with minimal guidance in an exploratory learning mode may also be undesirable, in terms of learning rate, for example ([29]; but cf. [22]). It is also not clear that very intricate skills could be "discovered" efficiently (compare inventing the calculus with learning it).

Clearly, this possibility interacts with the type of interface under consideration. Discovery learning is perhaps more feasible for a menu interface [22] than for a command interface: The former explicitly lists current options that can be tried and provides many implicit cues as well (e.g., in what the names of the options suggest). Interfaces designed around active forms [97, 98] provide even more implicit advice to the user about what can be done from a given system state. These and other techniques, such as dynamically highlighting and/or relocating currently important display areas, might be thought of as providing "soft" advice to the user.

Brown, Burton, and deKleer [10, p. 228] report that, in their experience developing interactive tutoring systems, providing large amounts of system-initiated intervention was quite often deleterious. They argue that it is often better to leave the user alone, especially if the problem seems small. However, they also suggest that no advice be given if the learner gets too far off track [12, p. 96], arguing that it is unclear what sort of advice to give in such cases anyway. This is probably the best available wisdom, but it underscores the lack of depth in our current understanding: What is too far off the track? What is a small problem? What is too much system-initiated intervention? These are empirical questions that await systematic investigation.

Consequences

Advice giving has consequences. At the very least, information is presented and hopefully imparted. The user can attend to it, be distracted by it, or ignore it. But other types of consequences are possible: The available function of the system can change when advice is given so as to implicitly compel the user to follow the advice, or the user could be placed in a special system mode where advice can be explored without risk of further errors or other complications.

Information. When a system "merely" provides information, it is providing a particular type and amount of information, timing its presentation in a particular way, etc. As we have seen in other cases, there is relatively little empirical work that has addressed the problem of fixing these parameters. Most of the relevant work consists of demonstration systems that adopt a certain specific set of parameters. There has, however, been some empirical study.

Lewis and Anderson [59] studied people learning to play a logic-based computer game and found that feedback immediately after an error was the most effective kind of advice. They also found that people learn to recognize dead-end situations better if they are allowed to encounter them. Allowing people to actually "see" the consequences of an error they would have made (and to easily back out of or avoid these consequences) is an important type of advisory information, and one well enabled by advice-giving systems. These findings suggest that advice should be presented immediately after an error is made, but that the user should also be allowed to see the consequence of the error before being allowed to correct it and go on (see also [75]).

Carroll and Kay [19] tested several versions of a system designed to teach the basics of word processing. One way in which the various versions differed was in the advisory dialogue: The control version, a commercial system, gave no advice. Other versions gave prompting advice, telling the user exactly what to do in the current system state to avoid making errors. Others gave feedback advice, telling the user how to recover when an error had been committed. One version gave both prompting advice, directing the user what to do, and feedback advice, directing the user how to recover from errors that were committed. Interestingly, the people who were trained on this latter version performed most poorly in transfer of learning tasks, suggesting (with Brown, Burton, and de Kleer [10]) that too much advice can be deleterious.

It seems intuitive that factors like the timing, amount, and type of advice will be important in designing advice-giving systems. These two studies support and develop this intuition. However, they are only two points in a vast space: Each involved other factors besides those we have highlighted in our discussion, and each addressed a different—and quite limited—task domain. Lewis and Anderson studied a simple game, and Carroll and Kay studied the elementary functions of a word-processing system. Behavioral work should be directed at these issues.

Confirmation Dialogue. Beyond merely presenting information, an advice-giving facility can modulate a system's control structure. For example, when the user can be diagnosed as having made an error or when the user is perhaps one command away from taking an action with potentially dangerous consequences, the system could enter into a "confirmation dialogue" by interrupting the session and posing questions that the user must answer before being permitted to continue with the session (e.g., "Type Y if the information you are about to transmit is nonproprietary, or anything else to cancel the command"). A well-known cliché example is the confirmation question, "Are you sure you want to delete all your files?" The objective, of course, is to force the user to rethink recent actions, immediate options, and possible consequences.

Many systems request confirmation for fatal errors (like "erase all files"). Cuff [28], in a review of work on database retrieval systems for casual users, advocates this sort of protective mechanism, but does not specify how or when it should be invoked. Gable and Page [38], also in a review of the literature, recommend a confirmation approach in which the user is given immediate feedback by the system that can be either heeded or rejected. But can we rest assured that a user who would erroneously issue such a command in the first place would adequately appreciate the severity of the consequences when they were briefly reviewed in a confirmation prompt? We need to know how the approach works and, perhaps more important, when and how it is likely to fail.

Control Blocking. In the confirmation approach the user's access to the system is temporarily interrupted by the advisory prompt. A more severe intervention is control blocking, where a portion of the system's function is rendered conditionally inaccessible to the user. Recall our example of the user who prematurely selected the "Application Customizing" function. If the user has no appropriate need for this function, it can be blocked off. The incorrect selection might then only elicit an advisory message to try something more appropriate. Similarly, if a user error were detected, the system could direct the user to correct the problem before allowing any other activity.

This approach has the undesirable side effect of profoundly interrupting the user. If the system has misdiagnosed the situation and no error has in fact been made, or if the user was doing something correct but unorthodox, the consequence would be extremely frustrating. The blocking approach may be most appropriate for new users whose goals may be more limited (and therefore more easily anticipated or recognized) and who may be less sensitive to interruptions in the flow of system control. Carroll and Carrithers [18] showed that an error-blocking approach led to more efficient learning of a word-processing application. No analogous work has yet been carried out with more experienced users.

Automatic Correction. Perhaps at the other extreme from confirmation dialogues and control blocking is automatic correction. In this approach, the system interprets a user error as a correct next response and allows progress to continue uninterrupted. Thus, we might imagine a user who misexecutes "split" and is automatically provided with a correct command form—without ever having to go through any explicit "help split" request. Interlisp's Do What I Mean (DWIM) facility [89] automatically resolves incorrect input and suggests a correction to the user (usually through nothing more sophisticated than spelling correction). In cases where the correction is classified as "obvious," though, the system simply makes the correction without requesting explicit confirmation. Clearly, this approach might be convenient for experienced and sophisticated users, particularly for mundane and common slips. However, the particular conditions under which the approach could be effective for such users have not been determined.

Two behavioral studies suggest that the automatic-correction approach may also have a use in the design of training interfaces for new users. We earlier described research by Carroll and Kay contrasting alternate versions of a training system [19]. The best training design therein evaluated, both from the standpoint of training time and success in a transfer of the learning task, was one with automatic correction. As the novice worked through a training scenario, errors were interpreted as the next correct action. Earlier work by Hillen [47] with an on-line manual reached the same conclusion. How far these curious findings can be extended is an open question (although it seems clear the approach must break down in the limit).

Protected Modes. All of the approaches we have considered provide advice within the context of

interaction with the system. An alternative is to create a special mode within which the user can work out problems and receive advice. Jagodzinsky [51] described a "reconnoiter mode" in which the actions of system commands are simulated without actually altering any of the user's data. A user who has a problem can switch into reconnoiter mode, resolve the problem by trying things out, and then return to the actual system environment to continue a work session. An idea like this was implemented in the SIGMA message processing service [81] and in a text-processing system called NLS-SCHOLAR, where the user, by asking a "What if . . . " question, creates a temporary copy of the current file and carries out the hypothetical action without affecting the actual workspace [46].

This approach leaves the responsibility for initiating a reconnoiter or what-if session to the user, but we might extend it to allow the system to suggest such advisory subsessions when potential user errors are detected. A problem is that, by placing the advisory subsession within a special mode, we risk confusing users by requiring them to keep track of different modes of operation [42, 68]. Clark [25] reported that users sometimes forgot the underlying concern that motivated a help request in the course of switching from problem-solving to learning mode. The growing availability of undo facilities suggests an interesting reconciliation of the utility for protected modes with the problem of mode changing: The actual system environment can become a protected environment for trying things out.

It seems likely that the bottom line regarding the management of initiatives and consequences in the design of advice-giving systems is that all of the approaches we have inventoried here (and no doubt the several we missed) can play useful roles. There is nothing wrong with this conclusion for the moment as long as we bear in mind how far it is from where we need to go if there is going to be a principled basis for designing such systems. We need to investigate the conditions under which each technique is most useful, and to understand the design trade-offs. Finally, we need to do these things empirically and systematically.

Scope

We have referred to error-recovery dialogues without considering what information should be provided in such dialogues. In DWIM automatic correction (e.g., [89]) no explicit information is provided, although information could be provided to help the user identify the specific error that triggered advisory intervention. The advice could help users to better understand what they are trying to do (in the sense of a goal), or to better understand how to do it (the concrete steps that facilitate accomplishing the goal), or perhaps to better understand the larger context within which this work and the particular task at hand are taking place.

Goals and Methods. Step-level advice can be presented very explicitly as literal directions. There is a degree of consensus that the focus of presented information should be on *how* to do something, in concise and direct instructions, rather than on more abstract or general explanations and descriptions [25]. Step-level advisory dialogue has also been advocated for intelligent advice-giving facilities [40, p. 152; 12, p. 92]. Burton and Brown state that they hoped to implicitly suggest appropriate goals to users through the language in which they presented appropriate steps.

One problem with this approach is that it magnifies the costs of any mistaken diagnosis the system might make. Users could easily be thrown far off the track if a system misdiagnoses their actual situations. A complementary problem arises if the suggested step (even though correct) is far from what the user expected. The user can take the suggestion, but without gaining any new understanding of the system. An alternative (reversing the strategy of Burton and Brown) would be to implicitly suggest steps by explicitly advising goals.

Edmonds [33], however, stressed that it can be unwise to specify very general-level objectives in detail because these goals will quite likely change as a function of the users' experience. For example, the vocabulary in which the user understands system-relevant goals may change radically with experience. Edmonds concluded that the means by which goals are attained should be specified first, since they are more stable. He specifically suggested that this step-level information could be presented by means of an example designed to facilitate analogical mapping to the user's current situation. The RABBIT system [97] allows the user to query a database through successive reformulation of an example target instance. However, presenting step-level advice through examples can also be problematic. It is notoriously difficult to differentiate those aspects of an example that generalize to an analogical domain from those that do not [48]. The advice may become ambiguous.

An alternative is to advise directly on goals. Particularly if the goal-level dialogue were couched at an intermediate level of abstraction, this approach might avoid the problems raised by Edmonds (it seems likely that the greatest changes in goal vocabulary as a function of user experience occur at the most abstract levels). Goal-level advice could in some cases also be more technically tractable than

step-level advice. Even if the system cannot determine exactly how the user ought to proceed, it might be able to delimit the options enough to pro vide goal-level advice. Indeed, this advisory strategy seems typical of what human advisers do when they remind learners explicitly of subgoals and keep track of where various results fit into higher goals [66]. While receiving advice on what goals to pursue—but not on specifically what to do or on particular functions and conditions for application—might be more of a learning demand on users, in the longer term it might also afford them a deeper understanding [4, 22].

An even more technically tractable approach might be to require the user to generate appropriate steps exclusively from implicit goal-level advice. The training wheels system [18] merely advised the user that a selected function was not available during training (the system blocked selected functions that were judged to have been prematurely selected, based on the user's experience and current task). This level of advice did not directly help the user to select an alternative. The user had to generate a new goal in view of the blocking message, and then derive the specific steps to achieve that goal.

Clearly, information and advice can be provided at a variety of different levels. A person could be advised exactly what step to carry out next, or could merely be given more information about appropriate goals. One approach does not exclude the other: A goal could be suggested explicitly when behavior (or an explicit help request) indicates that the user has lost the path in a complex procedure; conversely, a step could be suggested when the indication is that the user is pretty much on track but has made a small slip. Although some work has directly examined the types of strategies that are effective in human advice giving, there is much that we do not know. One aspect of the problem is that strategies for advice giving might need to be varied according to the needs of the user and the nature of the task.

Adjusting Advice. Jackson and Lafrere [50] urge that the type of advice provided should indeed vary as a function of the level of the task. Well-specified tasks may only require descriptive information support of the type found in many conventional help systems. Animated demonstrations of system function may be more appropriate for tasks requiring the integration of subprocedures, while even more open-ended tasks might require dynamically generated intelligent advice. Although Jackson and Lafrere cited no specific empirical backing or psychological reasoning for these suggestions, their view is consistent with the few behavioral studies of advice giving that have been published. McCoy [64] discussed anecdotes in which experts corrected misconceptions about the attribute mappings among objects. In her examples, the experts provided more information than was necessary to expose particular misconceptions; the additional information often served to establish the correct conception by raising questions about new issues.

McKendree et al. [66] studied a more procedural domain than McCoy: computer programming in Lisp. Advisor/learner diads worked through a series of typical programming problems in an unconstrained tutoring situation. The content of the advisory dialogue was analyzed in terms of the goals and methods employed by the advisors. This analysis showed that advisors tended to vary the content of their help according to a judgment of the "seriousness" of the misconception. A minor slip or syntactic error might elicit only enough information from the advisor to correct the immediate problem. However, a learner response indicating something more serious to the advisor might evoke more extensive explanation, examples, and questions. Again, though, it was not clear in this analysis just what information the advisors used to determine "seriousness."

Pollack [73] solicited electronic bulletin-board queries pertaining to a computer mail system. She noted that advice seekers do not always have well-formed plans about what they need to know. For example, they ask about actions that are impossible with respect to the system. A consequence was that advisors sometimes responded with an action not asked for—an alternate plan. Pollack directed these observations at the "appropriate query assumption" of Allen and Perault [2], the assumption that the user always has a well-formed plan and will ask an appropriate question. This result suggests that advice giving cannot presuppose the well formedness, with regard to a domain, of a request for advice.

Advising the Process. Several researchers have emphasized the importance of directing advice to *processes* involved in problem solving, rather than concentrating only on a solution to an immediate problem. This approach addresses a major deficiency in bug analyses of a user's knowledge and performance. Genesereth [40] argued that, in order to correctly analyze the bugs in a user's procedure, the entire problem-solving process, and not merely local segments (for example, the activity immediately preceding the manifestation of a bug), must be analyzed.

Fikes [35] dealt with procedural office tasks and the various levels at which these tasks can be described. He concluded that, although a rote description of steps may allow a worker to function in

many cases, it does not contain enough information to support problem solving in situations where specified results cannot be produced. For this, Fikes concludes the need for a "procedure teleology" containing a functional description of the task to be achieved and each step in the procedure. This allows users to recognize situations where a procedure might not be appropriate, and supports reformulation of new plans.

Coombs and Alty [26, 27] found that consultants often attended too little to the organization of the advisory process. Advisors assumed control of the dialogue as soon as possible and permitted only a one-way flow of information; they rarely included explanations of the larger problem context and rarely checked whether users really understood the advice. These problems appeared to vary according to the sophistication of the user, however. In advising relatively sophisticated users, advisors did tend to give explanations, and to share control of the advisory interaction to a greater extent. These latter interactions were viewed by users as being more successful.

These more successful advisory interactions often seemed to have little obvious structure; they seemed more rambling. More information seemed to be given than was really necessary. However, in debriefing sessions with users and consultants, motivations often became more apparent. The large information exchange served to make the inferences involved in problem solving explicit and allowed users to participate through monitoring and feedback. The end result was to encourage "problem solving through mutual understanding," with users working through problems with the help of the advisor. Coombs and Alty conclude that a guidance system should "support rather than direct problem solving, . . . mainly . . . [through] assistance in building and evaluating concepts in the problem area" [27, p. 27]. Thus, like Fikes, they conclude that advice that evolves with the user's knowledge about the processes, steps, and purpose of the task will hold the user in better stead when other problems are encountered (see also [54]).

Our analysis of advisory interactions between computer consultants and users also supports these conclusions. We found that ostensibly successful advisory interactions often did not codify a solution per se; rather the "solution" took the form of an enumeration of strategies and plans, along with an elimination of other courses of action judged to be less promising. We also found that one of the chief mechanisms through which advice is generated is the posing of verificational questions by the user: "All I do is read from standard input; is it that simple?" Thus, to an extent, "getting advice" merely means getting explicit encouragement for a solution one already knows (see [1, 65]).

Metacommunication. Although we have focused here on procedures, goals, and steps as levels for advice giving, it is important to bear in mind the larger context within which advice giving takes place. It is quite possible that the most important elements in effective advice lie beyond providing more correct and complete information. Alty and Coombs [3] made an empirical survey of the advice and information needs of users at a university computing center and found that one of the principal bases of user satisfaction with the center's help desk service was the personal human contact it provided. Similar points have been stressed by Eason [31, 32].

It is obvious that attitudes toward computers, particular systems, and particular advisory dialogues will influence users' ability to use and learn systems effectively. Burton and Brown [12, p. 89] provided positive encouragement as well as critique feedback in their WEST demonstrational system to help motivate users. However, such attempts to address motivational and attitudinal issues have so far rested on intuitive analyses of what these factors are and how they should be addressed. Malone [62] and Carroll [14] suggest that motivational elements used in computer games, such as fantasy, challenge, and safety in risk taking, might be incorporated into user interface dialogue designs to produce more intrinsically interesting system dialogues.

Summary

Current work on advice-giving systems has focused on system-initiated advice giving in the context of user error. This is probably an excellent choice; it is intuitively the type of situation where people might be receptive to advice and therefore one where we can get the leverage to demonstrate feasibility.

There are two main types of shortcomings in current research: First, the particular demonstration systems that have been developed and described in the literature have sampled the space of advisory interaction somewhat haphazardly. What level should advice be provided at—goals, steps, examples? What consequences should accompany advice—blocking, correction, special modes? How should advice be timed with respect to a triggering event such as an error? How much advice should be given? What dynamic aspects of advice-giving strategies are important?

A second class of shortcoming is that virtually none of the current work has been subjected to any systematic empirical study: Advice-giving systems *ought* to help people solve their problems, but we have no solid basis yet for knowing that they really

do. Indeed, Coombs and Alty [27] have recently posed the interesting question of why—now that demonstration advisory systems are available—this technology still has yet to be used much in real applications. They raise the possibility that this failure is due to a fundamental lack of usability. Similar points are made by Kidd and Cooper [54].

Behavioral evidence suggests that many forms of communication are used spontaneously in advice giving [1, 26, 66]. However, there have been no systematic investigations of the effectiveness of various styles or of the interaction with context or type of misunderstanding. Expertise in domain-specific problem solving and methods of teaching has been abstracted through experience—very limited experience in the case of automated systems. Advice-giving systems that have been built generally employ only one of these strategies or perhaps a few of them chosen randomly to vary advisory style. It is necessary to combine intuitions and behavioral studies to systematize more effective advising.

Advisory Dialogue Issues

Under what conditions should a system take the initiative in advisory dialogues?
Are there heuristics for advising error tangles that cannot be fully diagnosed?
How can we provide opportunities for discovery learning and yet still ensure adequate advisory support?
How can access to protected modes be engineered to avoid mode-changing usability problems?
When are obtrusive advice-giving approaches, like blocking and automatic correction, more appropriate than mere information presentation?
How can blocking be used with sophisticated users?
How can automatic correction be used for nontrivial problems?
At what level of problem detail should advisory dialogue be couched (e.g., should it advise goals or steps), and under what conditions?
How can we develop a more comprehensive theory of user goals in intelligent advice-giving systems?
In what presentation vocabulary should advisory dialogue be couched (e.g., examples, demonstrations, explanations, procedures)?
What are the specific usability benefits and costs of adjusting the content and presentation of advice dynamically?
How can we advise the problem-solving process *and not merely comment on the outcomes of that process?*
How can metacommunication in advisory dialogue be used to motivate users?

THE RESEARCH AGENDA

We have tried to capture and taxonomize some of the key issues in the design of advice-giving systems. We have not found a well-developed paradigmatic endeavor with a highly structured and generally agreed upon set of standard issues and a set of clear research successes. But this is not surprising. The area of advice giving is a frontier in both cognitive science and computer science.

In our view, the chief lessons to be drawn have to do with managing the research agenda. Too much energy is being directed at existence demonstrations—the design and implementation of limited-scale advice-giving technology. This is an appropriate early objective, and it is still relevant to investigate whether current small-scale advisory systems will scale up for large and complex, real applications. However, very little systematic effort has been directed at exploring the behavioral issues pertaining to advice-giving expert systems. This could be a critical deficiency, for even if current questions of technical possibility were to be settled with overwhelmingly impressive software, we would still be gambling that the type of advisory facilities developed would be usable by, and desirable to, real people.

To avoid this gamble two items need to be promoted on the research agenda: a psychological theory of advice and advising, and a behavioral methodology to assess the actual success of demonstration advice-giving systems.

Toward a Theory of Advice Giving

One approach to experimental computer science is to build a reasonable working system through which to explore, discover, and understand the principles of the system's operation. This is an important research method, and it has become important as rapid prototyping tools have become more powerful, flexible, and widely available. A limitation of this approach is that it is unconstrained in any area of system design with direct implications for the user interface. Moreover, it has become ever clearer that every aspect of system design has direct implications for the user interface. Accordingly, the approach of building experimental demonstration systems must be augmented to include usability considerations [20, 44].

The starting point must be a general understanding of what advice is, how people generate and deliver advice, and how people make use of advice in the context of problems. Much of our current under-

standing of advice and advice giving (like our understanding of usability in general) is either too high level or too low level. High-level principles like "advice should be relevant" or that they should "represent and communicate information about the system as people do" remind one of the vacuous maxim "know the user." But principles can also be codified at too low a level. Much current discussion of demonstration advice-giving systems offers design principles evaluated only casually in the context of a single system. Each of these systems is different in innumerable and unsystematic ways though, and they cannot really be compared, contrasted, or generalized with any confidence.

Useful principles will be applicable in a variety of domains and yet provide guidance for specific content and structure. One such principle might be "advice should be presented in a vocabulary of system-independent goals." This principle is strong enough to be falsifiable, in the Popperian sense, and to really constrain the design of advice facilities. Another principle that could be useful and that is empirically testable is "give immediate feedback when an error is detected." Exactly how this should be done is still a question. Should the user be blocked from any further progress or allowed to see consequences? We do not claim that these principles exemplify the best we can hope for, or even that they are correct (the evidence for them is still too limited). Rather, we suggest that they are the *least* we should settle for. Clearly, there is a lot of work to be done before we will have anything to seriously call a theory of advice, but at least studies are beginning to appear.

Measuring Advisory Effectiveness

Advice-giving systems are obviously intended for practical applications. The chief rationale for providing an advisory capability is to help users become more successful and effective. But, as we have tried to stress, good intentions are just not enough. Developing a theory of advice is a key to going beyond mere good intentions. Unfortunately, even when the problem is acknowledged, the solution offered can be useless (e.g., "implement . . . in a sensible way" [53] is not a methodology; it merely underscores the current lack of methodology).

We need to develop, codify, and routinize the use of behavioral evaluation techniques within the development process for advisory systems. We surely do not mean to suggest that there is a wealth of rich and powerful behavioral methodology that can now be taken off the shelf to improve advisory interfaces. There are in fact a variety of techniques available (some of which are referenced here; see also [6]). However, their effectiveness in helping to guide real, complex system projects is still something we have too little experience with to be able to confidently evaluate (but see recent case studies [7, 8, 15, 43]). Appropriate methods have to be constructed or adapted to this area.

One recommendation we would make is that researchers try to strike a balance between getting "hard" behavioral data that might be convenient to quantify and summarize, and "softer" behavioral data that might often be relatively more illuminating in helping to guide redesign (see [6, 7, 8, 15, 43]). Reiser, Anderson, and Farrell [75] focus considerable attention on behavioral evaluation and are able to report very encouraging overall statistics for the behavioral efficacy of the authors' Lisp tutor. However, they fail to report any qualitative details: For which error types did the tutor fail? How much of the power of the tutorial design was derived from the user-initiated *Clarify* and *Explain* keys, and how much from the system-initiated tutorial? We need to know, of course, that advisory designs can really work, and this can be assessed quantitatively. But it is the qualitative measurements that will guide further research developments [17].

Once we agree to measure—rather than merely to assert—the effectiveness of advice facilities, we can seriously ask whether a given advisory system is cost-effective with respect to any other. If one approach is 90 percent as effective as an alternative, but costs a tenth as much (in terms of development time and computing resources), it might be a very good deal. We referred earlier to the scenario machine approach, which encodes knowledge about a domain at a very large grain (that of a complete user action path). Such systems do not break knowledge down into procedural atoms and cannot generate inferences to respond dynamically, but their behavioral utility has already been systematically demonstrated. A scenario machine is cheap, but assessing its effectiveness will involve considerations of user and task specifics.

The research area of advice-giving expert systems is vital to a science of user interface design. The area must rest on a variety of technologies—knowledge engineering and dialogue management are two key ones we have looked at in this review. But success in designing expert systems for intelligent training and help will finally turn on usability. And, as we have argued, the area has not yet incorporated a serious psychological theory base or empirical methodology. Rectifying this will probably involve inventing psychological theory and method as much as assimilating existing theory and method. But the effort could be richly repaid: Advice giving could become the first successful domain for intelligent interfaces.

Acknowledgments. We thank Amy Aaronson, Robert Campbell, Richard Herder, Jakob Nielsen, Mary Beth Rosson, Ted Selker, Jeff Shrager, and John Thomas for comments on an earlier draft. Many of the thoughts incorporated into this article grew out of the discussions of a reading group we participated in that also included Amy Aaronson, John Black, Keith Bergendorff, Richard Catrambone, Nancy Grischkowsky, Richard Herder, Joeann Paige, Robert Mack, Jakob Nielsen, John Richards, Mary Beth Rosson, Linda Tetzlaff, John Thomas, and Ken Williamson.

REFERENCES

1. Aaronson, A.P., and Carroll, J.M. The answer is in the question: A protocol study of intelligent help. Res. Rep. RC 12034, IBM, Yorktown Heights, N.Y., 1986.
2. Allen, J.F., and Perault, C.R. Analyzing the intention of utterances. *Artif. Intell. 15* (1980), 143–178.
3. Alty, J.L., and Coombs, M.J. University computing advisory services: The study of the man–computer interface. *Softw. Pract. Exper. 10* (1980), 919–934.
4. Anderson, J.R. Cognitive psychology and intelligent tutoring. In *Proceedings of the Cognitive Science Society Conference* (Boulder, Colo., June 28–30). Cognitive Science Society, 1984, pp. 37–43.
5. Anderson, J.R., Farrell, R., and Sauers, R. Learning to program in Lisp. *Cognitive Sci. 8* (1984), 87–129.
6. Anderson, N.S., and Olson, J.R., Eds. Methods for designing software to fit human needs and capabilities. In *Proceedings of the National Research Council Workshop on Software Human Factors*. National Academy Press, Washington, D.C., 1985.
7. Bewley, W.L., Roberts, T.L., Schroit, D., and Verplank, W.L. Human factors testing in the design of Xerox's 8010 "Star" office workstation. In *Proceedings of CHI 83 Human Factors in Computing Systems* (Boston, Mass., Dec. 12–15). ACM, New York, 1983, pp. 72–77.
8. Boies, S.J., Gould, J.D., Levy, S., Richards, J.T., and Schoonard, J. The 1984 Olympic Message System—A case study in system design. Res. Rep. RC 11138, IBM, Yorktown Heights, N.Y., 1985.
9. Borenstein, N.S. The design and evaluation of on-line help systems. Doctoral dissertation, Tech. Rep. CMU-CS-85-151, Computer Science Dept., Carnegie-Mellon Univ., Pittsburgh, Pa., 1985.
10. Brown, J.S., Burton, R.R., and de Kleer, J. Pedagogical, natural language and knowledge engineering techniques in SOPHIE I, II, and III. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 227–282.
11. Burton, R.R. Diagnosing bugs in a simple procedural skill. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 157–183.
12. Burton, R., and Brown, J.S. An investigation of computer coaching for informal learning activities. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 79–98.
13. Card, S.J., Moran, T.P., and Newell, A. *The Psychology of Human–Computer Interaction*. Erlbaum, Hillsdale, N.J., 1983.
14. Carroll, J.M. The adventure of getting to know a computer. *Computer 15*, 11 (Nov. 1982), 49–58.
15. Carroll, J.M. Minimalist training. *Datamation 30*, 18 (Nov. 1984), 125–136.
16. Carroll, J.M. Satisfaction conditions for mental models. *Contemp. Psychol. 30*, 9 (1985), 693–695.
17. Carroll, J.M., and Campbell, R.L. Softening up hard science: Reply to Newell and Card. *Hum. Comput. Interaction*. To be published.
18. Carroll, J.M., and Carrithers, C. Training wheels in a user interface. *Commun. ACM 27*, 8 (Aug. 1984), 800–806.
19. Carroll, J.M., and Kay, D.S. Prompting, feedback, and error correction in the design of a Scenario Machine. In *Proceedings of CHI 85 Human Factors in Computing Systems* (San Francisco, Calif., Apr. 14–17). ACM, New York, 1985, pp. 149–153.
20. Carroll, J.M., and Rosson, M.B. Usability specifications as a tool in iterative development. In *Advances in Human–Computer Interaction*, H.R. Hartson, Ed. Ablex, Norwood, N.J., 1985, pp. 1–28.
21. Carroll, J.M., and Rosson, M.B. The paradox of the active user. In *Interfacing Thought: Cognitive Aspects of Human–Computer Interaction*, J.M. Carroll, Ed. Bradford Books/MIT Press, Cambridge, Mass. To be published.
22. Carroll, J.M., Mack, R.L., Lewis, C.H., Grischkowsky, N.L., and Robertson, S.R. Exploring exploring a word processor. *Hum. Comput. Interaction 1* (1985), 283–307.
23. Chin, D. An analysis of scripts generated in writing between users and computer consultants. In *Proceedings of the National Computer Conference*, vol. 53. (Las Vegas, July 9–12). AFIPS Press, Reston, Va., 1984, pp. 637–642.
24. Clancy, W.J. Tutoring rules for guiding a case method dialog. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 201–225.
25. Clark, I.A. Software simulation as a tool for usable product design. *IBM Syst. J. 20*, 3 (1981), 272–293.
26. Coombs, M.J., and Alty, J.L. Face-to-face guidance of university computer users—II: Characterizing advisory interactions. *Int. J. Man–Mach. Stud. 12* (1980), 407–429.
27. Coombs, M.J., and Alty, J.L. Expert systems: An alternate paradigm. *Int. J. Man–Mach. Stud. 20* (1984), 21–43.
28. Cuff, R.N. On casual users. *Int. J. Man–Mach. Stud. 12* (1980), 163–187.
29. Dearden, R.F. Instruction and learning by discovery. In *The Concept of Education*, R.S. Peters, Ed. Humanities Press, Atlantic Highlands, N.J., 1967.
30. Draper, S.W. The nature of expertise in UNIX. In *Human–Computer Interaction—Interact '84*, B. Shackel, Ed. North-Holland, Amsterdam, 1985, pp. 465–472.
31. Eason, K.D. The manager as a computer user. *Appl. Ergonomics 5* (1974), 9–14.
32. Eason, K.D. Dialogue design implications of task allocation between man and computer. *Ergonomics 23* (1980), 881–891.
33. Edmonds, E.A. Adaptive man–computer interfaces. In *Computing Skills and the User Interface*, M.J. Coombs and J.F. Alty, Eds. Academic Press, New York, 1981, pp. 389–426.
34. Elsom-Cook, M.T. Design considerations of an intelligent teaching system for programming languages. In *Human–Computer Interaction—Interact '84*, B. Shackel, Ed. North-Holland, Amsterdam, 1985, pp. 409–414.
35. Fikes, R.E. Automating the problem solving in procedural office work. In *Proceedings of AFIPS Office Automation Conference* (Houston, Tex., Mar. 23–25). AFIPS Press, Reston, Va., 1981, pp. 367–369.
36. Fischer, G., Lemke, A., and Schwab, T. Knowledge-based help systems. In *Proceedings of CHI 85 Human Factors in Computing Systems* (San Francisco, Calif., Apr. 14–17). ACM, New York, 1985, pp. 161–167.
37. Fitter, M. Towards more "natural" interactive systems. *Int. J. Man–Mach. Stud. 11* (1979), 339–350.
38. Gable, A., and Page, C.V. The use of artificial intelligence techniques in computer-assisted instruction: An overview. *Int. J. Man–Mach. Stud. 12* (1980), 259–282.
39. Gardner, H. *Frames of Mind: The Theory of Multiple Intelligences*. Basic Books, New York, 1983.
40. Genesereth, M.R. The role of plans in intelligent teaching systems. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 137–155.
41. Goldstein, I.P. The genetic graph: A representation for the evolution of procedural knowledge. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 51–77.
42. Good, M. Etude and the folklore of user interface design. *SIGPLAN Not. 16*, 6 (1981), 34–43.
43. Good, M.D., Whiteside, J.A., Wixon, D.R., and Jones, S.J. Building a user-derived interface. *Commun. ACM 27*, 10 (Oct. 1984), 1032–1043.
44. Gould, J.D., and Lewis, C. Designing for usability: Key principles and what designers think. *Commun. ACM 28*, 3 (Mar. 1985), 300–311.
45. Gould, J.D., Conti, J., and Hovanyecz, T. Composing letters with a simulated listening typewriter. *Commun. ACM 26*, 4 (Apr. 1983), 295–308.
46. Grignetti, M.C., Hausmann, C., and Gould, L. An "intelligent" on-line assistant and tutor—NLS-SCHOLAR. In *Proceedings of the National Computer Conference*, vol. 44 (Anaheim, Calif., May 19–22). AFIPS Press, Reston, Va., 1975, pp. 775–781.
47. Hillen, J.R.C. Comparison of four different self-paced manuals. Rep. HF047, IBM Hursley Human Factors Laboratory, Hursley Park, U.K., 1981.
48. Holyoak, K.J. Analogical thinking and human intelligence. In *Advances in the Psychology of Human Intelligence*, vol. 2, R.J. Sternberg, Ed. Erlbaum, Hillsdale, N.J., 1983.
49. Houghton, R.C., Jr. Online help systems: A conspectus. *Commun. ACM 27*, 2 (Feb. 1984), 126–133.
50. Jackson, P., and La Frere, P. On the application of rule-based techniques to the design of advice-giving systems. *Int. J. Man–Mach. Stud. 20* (1984), 63–86.
51. Jagodzinski, A.P. A theoretical basis for the representation of on-line computer systems to naive users. *Int. J. Man–Mach. Stud. 18* (1983), 215–252.

52. Johnson, W.L., Draper, S., and Soloway, E. Classifying bugs is a tricky business. In *Proceedings of the 7th Annual NASA/Goddard Workshop on Software Engineering* (Baltimore, Md., Dec. 1). NASA/Goddard, Greenbelt, Md., 1982.
53. Jones, K.S. Proposals for R&D in intelligent knowledge based systems. *J. Inf. Sci. 8* (1984), 139–147.
54. Kidd, A.L., and Cooper, M.B. Man–machine interface issues in the construction and use of an expert system. *Int. J. Man–Mach. Stud. 22* (1985), 91–102.
55. Kimball, R. A self-improving tutor for symbolic integration. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 283–307.
56. Klahr, D. Transition processes in cognitive development. In *Mechanisms in Cognitive Development*, R.J. Sternberg, Ed. Freeman, San Francisco, Calif., 1984.
57. Krauss, R.M., and Glucksberg, S. Social and nonsocial speech. *Sci. Am. 236* (1977), 100–105.
58. Ledgard, H., Whiteside, J.A., Singer, A., and Seymour, W. The natural language of interactive systems. *Commun. ACM 23*, 10 (Oct. 1980), 556–563.
59. Lewis, M.L., and Anderson, J.R. Discrimination of operator schemata in problem solving: Learning from examples. *Cognitive Psychol. 17* (1985), 26–65.
60. Mack, R.L., Lewis, C.H., and Carroll, J.M. Learning to use word processors: Problems and prospects. *ACM Trans. Off. Inf. Syst. 1*, 3 (July 1983), 254–271.
61. Malhotra, A., and Sheridan, P.S. Experimental determination of design requirements for a program explanation system. Res. Rep. RC 5831, IBM, Yorktown Heights, N.Y., 1976.
62. Malone, T.W. Toward a theory of intrinsically motivating instruction. *Cognitive Sci. 4* (1981), 333–369.
63. Mantei, M., and Haskell, N. Autobiography of a first-time discretionary microcomputer user. In *Proceedings of CHI 83 Human Factors in Computing Systems* (Boston, Mass., Dec. 12–15). ACM, New York, 1983, pp. 286–290.
64. McCoy, K.F. Correcting misconceptions: What to say when the user is mistaken. In *Proceedings of CHI 83 Human Factors in Computing Systems* (Boston, Mass., Dec. 12–15). ACM, New York, 1983, pp. 197–201.
65. McKendree, J., and Carroll, J.M. Advising roles of a computer consultant. In *Proceedings of CHI 86 Human Factors in Computing Systems* (Boston, Mass., Apr. 13–17). ACM, New York, 1986, pp. 35–40.
66. McKendree, J., Reiser, B.J., and Anderson, J.R. Tutoring goals and strategies in the instruction of programming skills. In *Proceedings of the Cognitive Science Society Conference* (Boulder, Colo., June 28–30). Cognitive Science Society, 1984, pp. 252–254.
67. McKendree, J., Schorno, S., and Carroll, J.M. Personal Planner: The Scenario Machine as a research tool. *SIGGRAPH Video Rev. 18* (1985). (Video demonstration shown at *CHI 85 Human Factors in Computing Systems Conference*, San Francisco, Calif., Apr. 14–17.)
68. Newman, M., and Sproull, F. *Principles of Interactive Computer Graphics.* 2nd ed. McGraw-Hill, New York, 1979.
69. Nielsen, J., Mack, R.L., Bergendorff, K., and Grischkowsky, N.L. Integrated software usage in the professional work environment: Evidence from questionnaires and interviews. In *Proceedings of CHI 86 Human Factors in Computing Systems* (Boston, Mass., Apr. 13–17). ACM, New York, 1986, pp. 162–167.
70. Nisbett, R., and Wilson, T. Telling more than we can know: Verbal reports on mental processes. *Psychol. Rev. 84* (1977), 231–259.
71. Norman, D.A. Categorization of action slips. *Psychol. Rev. 88* (1981), 1–15.
72. Papert, S. The role of artificial intelligence in psychology. In *Language and Learning: The Debate between Jean Piaget and Noam Chomsky*, M. Piattelli-Palmarini, Ed. Harvard University Press, Cambridge, Mass., 1980, pp. 90–99.
73. Pollack, M.E. Information sought and information provided: An empirical study of user/expert dialogues. In *Proceedings of CHI 85 Human Factors in Computing Systems* (San Francisco, Calif., Apr. 14–17). ACM, New York, 1985, pp. 155–159.
74. Pope, B. A study of where users spend their time using VM/CMS. Res. Rep. RC 10953, IBM, Yorktown Heights, N.Y., 1985.
75. Reiser, B.J., Anderson, J.R., and Farrell, R.G. Dynamic student modelling in an intelligent tutor for Lisp programming. In *Proceedings of IJCAI* (Los Angeles, Calif., Aug. 18–23). Kaufman, Los Altos, Calif., 1985, pp. 8–14.
76. Rich, E. A. Programs as data for their help systems. In *Proceedings of the National Computer Conference*, vol. 51 (Houston, Tex., June 7–10). AFIPS Press, Reston, Va., 1982, pp. 481–485.
77. Rich, E. Users are individuals: Individualizing user models. *Int. J. Man–Mach. Stud. 18* (1983), 199–214.
78. Rosson, M.B. Patterns of experience in text editing. In *Proceedings of CHI 83 Human Factors in Computing Systems* (Boston, Mass., Dec. 12–15). ACM, New York, 1983, pp. 171–175.
79. Rosson, M.B. The role of experience in editing. In *Human-Computer Interaction—Interact '84*, B. Shackel, Ed. North-Holland, Amsterdam, 1985, pp. 45–50.
80. Rosson, M.B., and Mellen, N.M. Behavioral issues in speech-based remote information retrieval. In *Proceedings of AVIOS 85*, L. Lermon, Ed. American Voice I/O Society, San Francisco, Calif., 1985, pp. 23–34.
81. Rothenberg, J. On-line tutorials and documentation for the SIGMA message service. In *Proceedings of the National Computer Conference*, vol. 48 (New York, June 4–7). AFIPS Press, Reston, Va., 1979, pp. 863–867.
82. Rumelhart, D.E., and Norman, D.A. Representation in memory. In *Steven's Handbook of Psychology*, R.C. Atkinson, R.J., Hernstein, G. Lindzey, and R.D. Luce, Eds. Wiley, New York, 1986.
83. Shrager, J., and Finin, T. An expert system that volunteers advice. In *Proceedings of the National Conference on Artificial Intelligence* (Aug. 18–20). AAAI, Pittsburgh, Pa., 1982, pp. 339–340.
84. Shulman, L.S., and Keislar, E.R., Eds. *Learning by Discovery: A Critical Appraisal.* McNally, Chicago, Ill., 1966.
85. Simon, H.A. The functional equivalence of problem solving skills. *Cognitive Psychol. 7* (1975), 268–288.
86. Sleeman, D. Assessing aspects of competence in basic algebra. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 185–199.
87. Sleeman, D., and Brown, J.S., Eds. *Intelligent Tutoring Systems.* Academic Press, New York, 1982.
88. Stevens, A., Collins, A., and Goldin, S.E. Misconceptions in students' understanding. In *Intelligent Tutoring Systems*, D. Sleeman and J.S. Brown, Eds. Academic Press, New York, 1982, pp. 13–24.
89. Teitelman, W., and Masinter, L. The Interlisp programming environment. *Computer 14*, 4 (Apr. 1981), 25–34.
90. Thomas, J.C. A method for studying natural language dialog. Res. Rep. RC 5882, IBM, Yorktown Heights, N.Y., 1976.
91. Thorndike, E.L. The psychology of learning. In *Educational Psychology*, vol. 11. Teachers College of Columbia University, New York, 1913.
92. Turiel, E., and Davidson, P. Heterogeneity, inconsistency, and asynchrony in the development of cognitive structures. In *Stage and Structure: Reopening the Debate*, I. Levin, Ed. Ablex, Norwood, N.J., 1986, pp. 106–143.
93. VanLehn, K. Bugs are not enough: Empirical studies of bugs, impasses and repairs in procedural skills. Tech. Rep. CIS-11, Xerox PARC, Palo Alto, Calif., 1981.
94. VanLehn, K. Felicity conditions for human skill acquisition: Validating an AI-based theory. Tech. Rep. CIS-21, Xerox PARC, Palo Alto, Calif., 1983.
95. Weizenbaum, J. ELIZA—A computer program for the study of natural language communication between man and machines. *Commun. ACM 9*, 1 (Jan. 1966), 36–45.
96. Wilensky, R., Arens, Y., and Chin, D. Talking to UNIX in English: An overview of UC. *Commun. ACM 27*, 6 (June 1984), 574–593.
97. Williams, M.D. What makes RABBIT run? *Int. J. Man–Mach. Stud. 21* (1984), 333–352.
98. Zloof, M. Query-by-example: A data base language. *IBM Syst. J. 16* (1977), 324–343.

CR Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: User/Machine Systems—*human factors*; I.2.1 [**Artificial Intelligence**]: Applications and Expert Systems

General Terms: Documentation, Human Factors

Additional Key Words and Phrases: Help, human–computer interaction, usability, user support

Authors' Present Addresses: John M. Carroll, User Interface Institute, IBM Watson Research Center, Yorktown Heights, NY 10598; Jean McKendree, Dept. of Psychology, Carnegie-Mellon University, Pittsburgh, PA 15213.