On Interface Requirements for Expert Systems

Richard L. Wexelblat

The user interface to an expert system shares many design objectives and methods with the interface to a computer system of any sort. Nevertheless, significant aspects of behavior and user expectation are peculiar to expert systems and their users. These considerations are discussed here with examples from an actual system. Guidelines for the behavior of expert systems and the responsibility of designers to their users are proposed. Simplicity is highly recommended. Entia non sunt multiplicanda praeter necessitatem.[1]

The user interface to a software system can spell the difference between success and failure. Sometimes, function does not seem to count. If the program does a good enough job, if the users see an easy to use, easy to learn, helpful, pleasant interface, they love it. The interface might be the most significant sales aspect of a software product (consider the spate of look-and-feel lawsuits!). This wasn't always the situation. In the early days of online computing, users were so happy to have something that they accepted almost anything. Many of today's AI systems implicitly exploit this phenomenon. The novelty of the expert system shelters poor human factors (as does the fact that many users of today's expert systems have been intimately involved in their creation). It is still the case in some—though not all—application domains that people are so pleased to have a working expert system they accept aggravation in its learning and use. The media hype that surrounds AI in any form has added snob value to having an expert system. As the novelty continues to wear off and the expert system loses its snob value, normal measures of usability will apply. Developers who do an inadequate job of design will have less than satisfactory acceptance.

Users have implicit models of a system and designers must acknowledge these subjective models in the interface design. If the system is to behave properly across a range of users, the system should at least implicitly contain a model of its users. The user's model will define what the system is expected to do in terms of user expectations. The system's user model, first by anticipating and later by recognizing the knowledge and motivation in the user, will be able to respond appropriately to perceived levels of user ability. Early user models in the literature tend to cite a one-dimensional range of user expertise, from beginner to expert (Shneiderman 1979; Schneider, Wexelblat, and Jende 1979). Such a model is simplistic in that after some experience, users tend to be expert in some features and novice in others. Casual or occasional users are difficult to include in a model. They tend to move from (apparently) novice to expert level in a few interactions, then repeat this process the next session.

With expert systems, problems of user modeling and interface design are compounded. A user can be experienced in the problem domain but naive with respect to the system or computers in general. Domain knowledge can be subjective, based on the expertise of a single expert. It can even contain subtle inconsistencies, reflecting valid differences of opinion among experts. An expert system tends to be brittle: In a limited domain of applicability it works well, but outside the domain it doesn't work at all. It is difficult for the user—expert or novice—to keep the bounds of the domain in mind at all times. In fact, users expect more from an

0738-4602/89/$3.50 ©1989 AAAI

expert system than from a conventional computer program, but their expectations are harder to model. By its nature, an expert system has fewer constraints on input and output, and users tend to be unsure of the limits; so, more sorts of user assistance are needed. Furthermore, the domain of applicability and the limits on this domain are harder for a user to understand.

Because the nature of an expert system is to allow (controlled) imprecision, it is impossible to take advantage of the sorts of data or syntax checking that one would find in, say, a payroll program or BASIC interpreter. Moreover, the nature of an expert system leads to the need for more than one interface. Different classes of users require significantly different sorts of interactions.

Interfaces to expert systems are beginning to be addressed (Carroll and McKendree 1987; British Computer Society Workshop on People and Computers: Designing the Interface, Cambridge, U.K., 17–20 September 1985), but the current literature tends to be anecdotal in nature and to reflect subjective value judgments. Unfortunately, little hard data exist to confirm the body of opinion in interface design. This article is similar in nature but reflects my experience of more than three decades of designing and thinking about user interfaces. It represents as many years of working with computers, mostly in positions where I had little control over the interface. Although subjective in approach, this article attempts to discuss criteria, models, and guidelines in a way that can be understood and applied by system designers who do not necessarily share my definitions. Where opinion is backed by evidence, details are cited. For the rest, I hope that the opinion appeals to designers of future systems. Those interested in surveying the literature should begin with Ben Shneiderman's (1987) latest book. Other helpful material can be found in the references.

How Are Expert Systems Used?

An expert system is a computer program. As with any computer program, it can be applied where appropriate. Consider an expert system for the diagnosis and repair of electronic equipment. It can run on a large central computer, and technicians in the field can call in to ask questions of an operator when a difficult problem arises. At the other extreme, the diagnostic system can be embedded in the equipment with its own processor, monitoring the equipment as it runs, diagnosing problems when it fails. Another scenario for diagnosis is the portable system carried into the field and plugged into the target equipment for preventive or diagnostic maintenance.

The media hype that surrounds AI in any form has added snob value to having an expert system.

The design of the user interface for an expert system will depend on the operating environment and the qualifications of the user. It is necessary to choose the computer on which to implement the diagnostic system and to see if signals can be read automatically from the target equipment or if a human is needed to transfer the information. Although modern electronic systems are usually instrumented for automatic reading of internal values, mechanical and optical systems or subsystems typically won't have this ability. An amazing number of free variables relate to diagnostic expert systems. For example, the system to be diagnosed can be uninstrumented, partly instrumented, or fully instrumented. The diagnostic system can operate autonomously or with an operator or an external control computer. It can be implemented on a mainframe, workstation, portable computer, or hybrid (for example, personal computer´modem´workstation). The diagnostic system can be embedded in equipment under test, on site plugged into equipment under test, on site as a stand-alone system, portable carried to the site, remote connected to equipment under test through a modem or network, or remote as a stand-alone system.

Expert systems today are implemented in almost all combinations of these modes. At least for the next few years, the advantage of the power of the mainframe must be balanced against the convenience of portability. In diagnosis, for example, the graphics capability of a (non-portable) workstation must be weighed against the value of an embedded capability. The difference between the workstation and the personal computer is not completely clear today, and the boundary—if any—is becoming more vague. For purposes of this discussion, differentiation is made between workstation and portable computer. Portable does not, however, imply lightweight. Until a powerful, lightweight computer with good gray-scale graphics

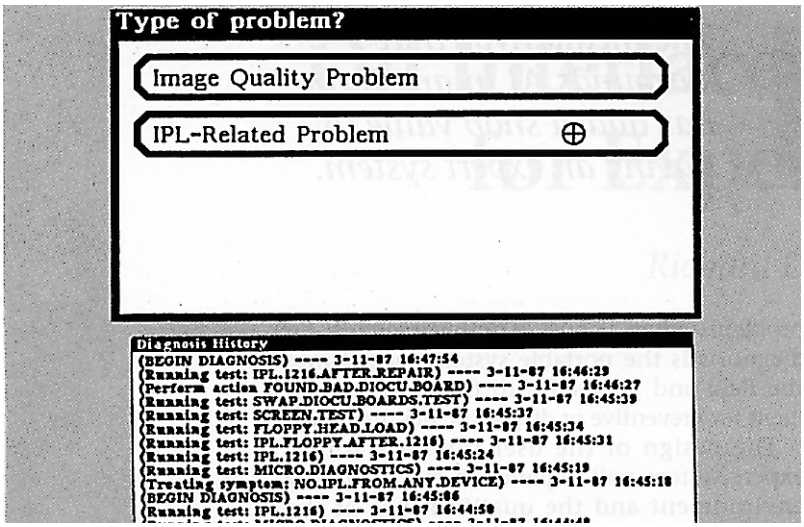

Figure 1. Top-Level Screen for Tomoscan Diagnostic Program.

becomes available, hybrid or remote configurations are likely for systems not powerful enough to contain their own diagnostics.

An Actual System and Its Interface

To make the discussion of interfaces tangible, examples are taken from a specific development. The knowledge engineering group at Philips Labs in Briarcliff, New York, working with the service department of Philips Medical Systems in Eindhoven, Holland, built a development environment for diagnostic systems called CATS (Lee et al. 1988). "CATS" stands for the objects in the knowledge base—components, actions, tests, and symptoms. Its initial application was for field diagnosis of the Philips Tomoscan 300 series of tomographic scanners. The examples here relate to the interface of the delivered expert system, not to the knowledge engineer's interface, which is a different and more complicated problem. The problem treated in the sample screens is a dead control computer, or one that will not load.

This particular application is oriented toward a novice user, who, although an experienced technician, is not expected to know much about the Tomoscan itself. Operation is strictly mouse-menu-, and window-oriented, and all communication is through single-function windows. That is, a given window contains only one type of information. Each sort of information has its particular window; this window occupies a specific location on the screen, although the size can vary depending on the contents. An unused window does not appear, and temporary windows can cover parts of other windows. Avoiding use of the keyboard was a deliberate design decision, although a few exceptions, such as having the user enter a name or identification at the beginning of the session, do exist. The sample screens were chosen to illustrate aspects of the user interface rather than the totality of the system's abilities. (Mixed-initiative interaction, where the user can volunteer information, break the decision tree flow, and disagree with system hypotheses, is possible in a version of CATS currently being developed, but user interface considerations have not yet been completely worked out.)

The initial screen during a CATS run consists of a logo and a single button requesting the user to "Click Here to START DIAGNOSIS." The basic user operation, pointing and clicking, is referred to as selecting. Once the run is begun, the screen in figure 1 appears. The upper right interaction box lists in menu form the classes of problems appropriate to the current situation. The circled + symbol is the cursor. Elsewhere in the run, this window is used for presenting questions and lists of symptoms. In this example, only two top-level problems are included. The other window initially appearing (at the lower right) is a history trace. Currently, history consists of an audit trail of interactions. Eventually, it will provide a structured record of the interaction in terms meaningful to the user.

When the user selects "IPL-Related Problem," the screen illustrated in figure 2 appears, showing two potential problems in the interaction box. This screen both implies and presumes that this set of problems is exactly what the user needs to handle. The user is entitled to assume that these problems are exactly the set of things which can go wrong with the initial program loading (IPL). Of course, these symptoms are necessarily the ones the domain expert told the knowledge engineer about. No symptoms are present that the domain expert neglected to mention. Conditions deemed by the domain expert to be unlikely to occur are also missing. These conditions are sometimes referred to as *boundary knowledge* and handling them remains an open research issue. The completeness of the knowledge base is beyond the scope of this article. Nevertheless, it is as critical to the success of the system as any part of the user interface and is addressed in the CATS knowledge engineering tool set, which treats knowledge base correctness, consistency, and completeness.

In addition to the new symptoms in the right-hand window, the entire left side of the screen is filled with an array of small boxes labeled with the names of functional components. At first glance, this display seems over complicated, but it actually represents the physical structure of the device's control computer: its backplane and power section. An experienced technician identifies with this representation. The field trial indicated that even a neophyte is at ease with such a representation soon after noting the conceptual mapping onto the physical device. The components are marked UNTESTED. These labels are updated as the diagnosis proceeds.

When appropriate, How and Why buttons appear. Figure 3 shows what happens when the user asks how. In addition to the explanation box, a graphic appears illustrating the switches referred to in the original query. When no graphic is appropriate to the explanation, no picture box appears. If no how or why information is appropriate to a query, the corresponding button is omitted from the display. Both the picture and explanation boxes disappear when the user indicates no further need for the information.

Figure 4 shows the state of the display after successful diagnosis and repair. The technician has replaced the 1216 disk controller (in the middle of the left-hand column). The replacement was tested and found to fix the problem. The label on the 1216 disk controller changed from UNTESTED to BAD to NEW. UNTESTED to NEW.OK as the interaction progressed. Note that the completion message in the interaction box restates the problem CATS believes it solved. In a lengthy interaction with many side trips, the user might well forget what the original problem was.

In this figure, several of the component boxes relating to power distribution contain OK. Although power was not explicitly involved in the interaction, the user's responses indicated behavior that would not have been possible without power. The system knows about power distribution and deduced the presence of power and the state of at least part of the power-distribution subsystem. The 1216 disk controller is powered from the top power supply, now marked OK. The bottom power supply is not involved and remains UNTESTED.

Figure 5 is a screen from a test of the optical subsystem. At this stage in the diagnosis, a representation of the physical machine is not appropriate, and the left half of the screen is blank. When How information is requested, however, the explanation and graphics boxes still appear in the same relative positions. In this example, the graphic is a representation of the image the technician is expected to see.

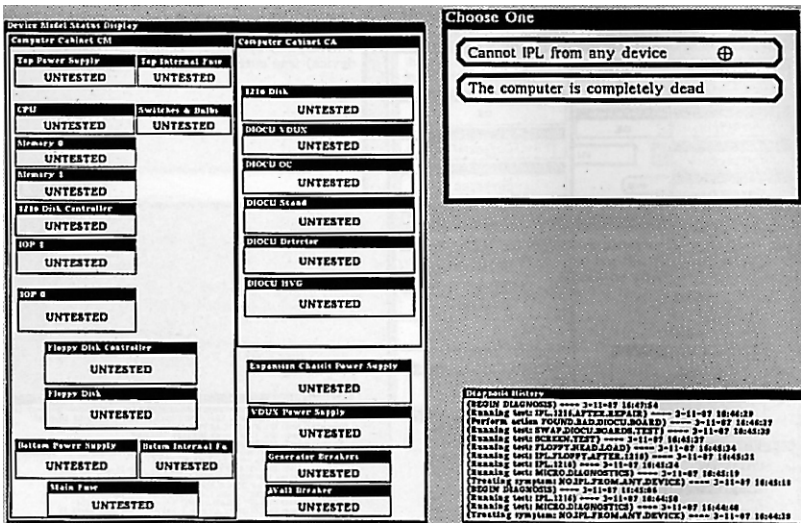

Figure 2. Diagnostic Screen for Tomoscan Electronic Subsystem.

Users and Expert Systems

The Tomoscan diagnostic demonstration is an example of good systems design and potentially excellent user interface design. (The terms good and excellent are highly subjective. My own definitions and opinions are documented elsewhere [Wexelblat 1981, 1983].) What, however, in this implementation is uniquely relevant to diagnosis expert systems? An easy answer is "nothing." A system is a system is a system. No reason exists for an expert system to be any different

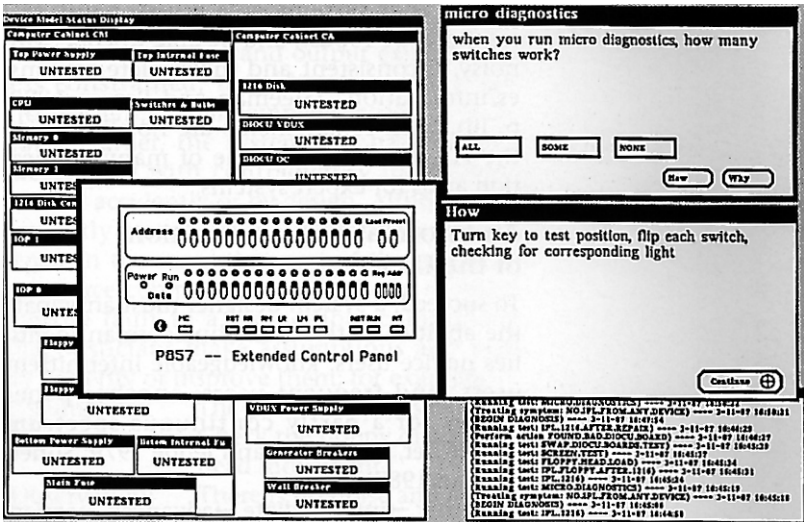

Figure 3. Tomoscan Diagnosis: The How Information.

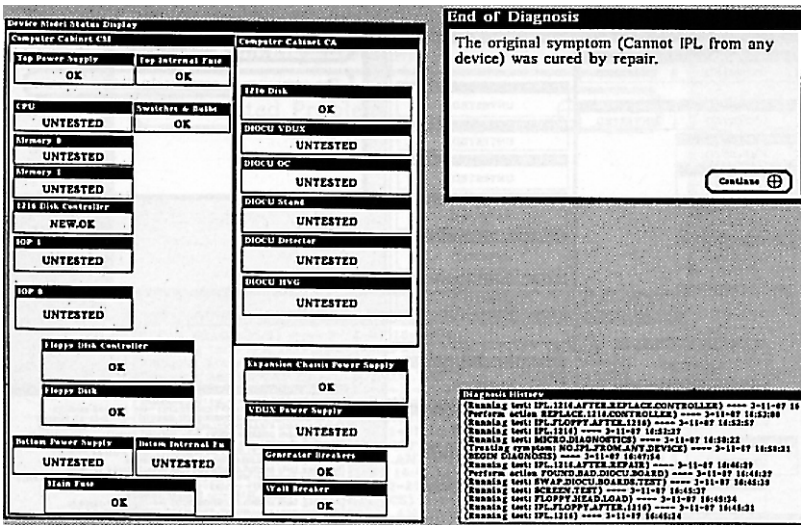

Figure 4. Tomoscan Diagnosis: Successful Session.

. . . would this were so. A difference does exist between an expert system and, say, a computer-aided design program, a data-base, a spreadsheet, or even an expert system shell. We expect a beginner to spend weeks or even months learning to use a Lotus, SPICE, or KEE. However, circumstances are quite different with an expert system. The user of an expert system expects to begin useful work almost immediately. The difference comes from the user's knowledge; the user's behavior; the user's training; and, especially, the user's expectations. The difference is also in the kinds of interactions, the kinds of data the user enters, and the computer's responses.

In the introduction to a book on the design of intelligent tutoring systems (ITSs) it is noted, "That new AI techniques have been evolved is not surprising given the demands of ITS; that systems give a reasonably fast response, be robust, and be able to cope with noisy, inconsistent and incomplete responses/information" (Sleeman and Brown 1982, p. iii). Such requirements are not limited to the ITS domain, only one of many application areas for expert systems.

An Informal Characterization of the User

To succeed, a system designer must anticipate the abilities of the user. Shneiderman identifies novice users, knowledgeable intermittent users, and frequent users; another paper argues for a partly continuous spectrum (Schneider, Wexelblat, and Jende 1979; Shneiderman 1987):

beginner →intermediate →advanced →expert

Both are valid models. Users can smoothly progress from beginner to expert, but some stall at levels in between. A large group of infrequent users also displays a mixed expertise, one that shifts suddenly from level to level. With an expert system, the spectrum of ability spans at least four dimensions: (1) the domain knowledge of the user, (2) the user's ability to solve problems in the domain without an expert system, (3) the capability and scope of the expert system, and (4) the host hardware and software.

(An independent factor is the user's familiarity with computers and comfort in using them: During the first CATS field trial, one subject was sure the computer was sitting in judgment, grading his ability to service Tomoscans. He was so fearful, he refused several times to select an "I don't know" button that would have led him through additional menus. The cause of this problem was eventually traced to improper explanation of the trial circumstances, but this sort of abstract problem is almost impossible to anticipate.)

The first two previous points, although related, are not identical. It might be useful for a television repairperson to be an electronics expert, but such expertise is neither necessary nor sufficient. Another dimension is added to the spectrum by the intent of the expert system itself. If it is an adviser, the user must presumably know a good bit about the domain. If it is the expert, the user need not know much at all. Yet another aspect is the "who's in charge around here" phenomenon, the touchy balance between task automation and the user's autonomy. A system that takes charge and controls the interaction might be ideal for one user in one domain but totally unacceptable to a user with another level of expertise. An overly pushy system will be pushed aside.

The designer's understanding of these dimensions contributes to usability and, hence, to user satisfaction and system acceptance. In theory, an expert system shell should provide the interface so that the implementor only needs to worry about function. This excellent concept is not yet realized in a commercial system. Here, however, is a proposed rule 1 of expert system design:

Make the behavior of the system and its host hardware so easy to learn and use that user satisfaction depends solely on the success of the knowledge engineering.

As noted earlier, the CATS example uses only the screen for output and primarily the mouse for input. For someone at all familiar with terminals or computers, to learn the hardware takes seconds. Displays are legible and consistent in appearance and behavior,

simple where possible, modeling the domain where complex. Fancy fonts and non-functional ornamentation are avoided. User training is measured in tens of minutes rather than hours or even days. The user pays attention to the problems of the domain, not the learning of the system. The diagnostic structure and even the vocabulary used in the system contain implied assumptions of what users will understand. Overconcern with learning is dangerous. Without going into detail, it is appropriate to note one of the basic rules for the design of systems of any sort: Ease of learning is not the same as ease of use. (In fact, they might be inversely related. Easy to learn might lead to harder to use in the long run.)

Implied promises

Users expect more from an expert system than from a conventional computer program, but their expectations are harder to understand, codify, and model. The nature of an expert system is to reason and give answers that might well not be obvious to the user. By appearing intelligent, an expert system can unintentionally seduce a user into believing that it is intelligent. Even though the user knows the computer is a machine, the more it does, the more intelligent it appears, and the more users can be misled (or can mislead themselves). This situation has existed for years—perhaps since the non-English English-like Cobol programming language was first released. Cobol's superficial resemblance to English is misleading. DIVIDE BALANCE INTO 3 means

$$\frac{3}{\text{BALANCE}}$$

not

$$\frac{\text{BALANCE}}{3}$$

Books have been written about the tendency of people to endow the early conversational system ELIZA, essentially a dumb program, with intelligence and natural language capability (Weizenbaum 1976). To coin a cliché, a little intelligence is a dangerous thing. A computer program is a fully deterministic, finite-state process. A tendency exists, enhanced by the overuse of anthropomorphisms (as I did earlier when I wrote, "CATS believes it solved"), to create the image of a program as a dumb human, a sure way to mislead the user's expectations.

The expert system must clearly state its limits, which is difficult to do in general, but necessary in any particular application. As long as the system handles part of the problem, users expect it to handle the whole problem. If it solves the whole problem, people expect it to solve more problems. During early demonstrations of Tomoscan diagnosis, it was hard to keep visitors focused on the matter at hand: a computer was actually diagnosing problems with a scanner. A typical interaction follows:

Demonstrator: At present only one symptom is included for the optical subsystem. Others will be added before the field trial begins.

Visitor: I understand completely. Tell me, why is there no ability to handle {long list of other optical subsystem problems}?

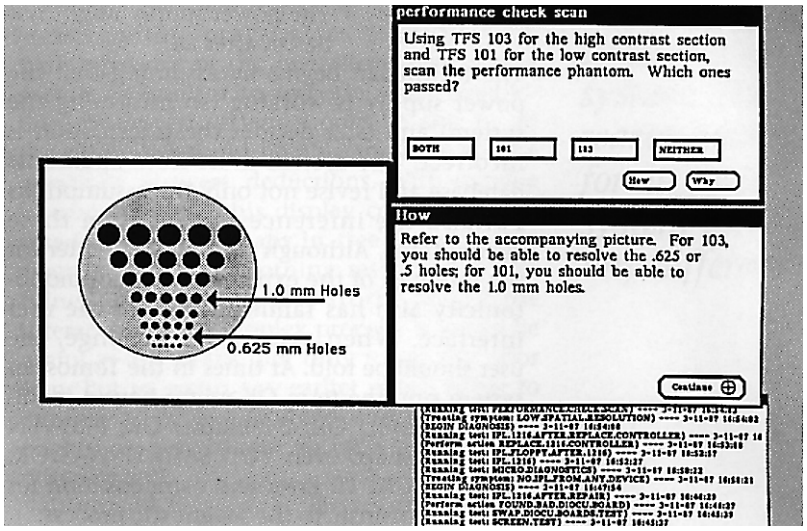

Figure 5. Tomoscan Diagnosis: Optical Subsystem.

Input and Output

By its nature, an expert system has more work to do in input verification and error checking. More kinds of input and output exist and are less constrained. In addition to the noisy, inconsistent, and incomplete responses mentioned earlier, the system will have to cope successfully with contradictory information entered accidently or by design. Although not explicitly stated earlier, diagnosis embraces repair in the sense that part of the process is to replace components in the hope of isolating the fault to a replaceable component. Diagnosticians make assumptions and then try to verify or disprove them; for example:

Assumption:	The power supply is OK.
Test:	Check the floppy disk drive head movement.
Observation:	There is no noise and no motion, and the pilot light is off.

Assumption:	The power supply might not be OK after all.

The technician begins by assuming that the power supply is working (so informing the system) and later decides the information is incorrect. The system must now update its database and revise not only the assumptions but also the inferences drawn from these assumptions. Although primarily a matter for the internals of the expert system, nonmonotonicity also has ramifications for the user interface. When assumptions change, the user should be told. At times in the Tomoscan system run, the term OK might have a modifier ASSUMED OK; INFERRED OK; PROVEN OK; or perhaps even YOU SAID IT WAS OK, BUT I DOUBT IT. Error and compensation for error are intrinsic to the diagnostic process:

Observation:	There are no lights, no nothing.
Assumption:	There is a power problem.
Observation:	The fuse is blown.
Action:	Replace it. (Other diagnostic activities until suddenly:)
Observation:	There are no lights, no nothing.

In fact, the blown fuse was a symptom rather than the cause of the problem. Fixing the fuse connects power to the rest of the system, but eventually overheating somewhere causes the fuse to blow again. Two functional and interface considerations apply here. The replaced fuse should be marked REPLACED or UNTESTED. Also, the work done between replacing the fuse and its blowing out should not be wasted. Even if the technician has to divert attention to the power section, it must be convenient to return afterwards to the point where the fuse blew. The interface must make available information sufficient to establish the prior context. Further, tools must be available to enable and assist many levels of context shifting. In the evolving mixed-initiative interface of CATS, an entire new class of problems has arisen. The user volunteers information that might (usually much later in the session) conflict with system-deduced hypotheses. By definition of the resolution strategy, it is the responsibility of the user to decide which assertion to retract, and it is the responsibility of the system to display the conflict and the support for each alternative in domain-specific terms adapted to the interaction. That is, rather than just say, "You said A and CATS deduced not-A," an entire context must be set. Furthermore, after resolution (which can entail several additional test or repair actions), the session must be placed back into a situation meaningful to the user. Although the user interface aspects of this sort of interaction are only a small part of what is needed, form is as important as function.

Help

Earlier it was noted that more sorts of user assistance are needed. One reason is that the expert system, especially if capable of reasoning and inference, is working by methods not likely to be understood by the user. Another reason is that it is difficult for the user to understand fully the domain of applicability and the limits on the domain. At least six distinct sorts of help information exist that the user of an expert system might require. They are characterized by the following questions the user might ask of the expert system and are discussed in the subsequent discussion:

How do I do what you ask me to do?
Why do you ask me to do this task?
How did you come to this question or conclusion?
By what steps did we get here?
What do I do next?
What do you know about? Or even worse:
Do you know about X?
What can you tell me about X?
Can I do Y?

How do I do what you ask me to do? The user was asked to perform some action and needs instruction on how to achieve it. The system must respond in terms meaningful to the interaction. Examples of such interactions appear in figures 3 and 5. Of course, in a system designed for users with different levels of domain expertise, different sorts of responses are required.

Why do you ask me to do this task? In other words, what is the purpose of the action you suggest. If the user selects Why in response to "when you run diagnostics, how many switches work," the system responds, "To test the CPU switches and bulbs." Such a query is more than just an idle question. A novice might use it to learn more about the domain, or an expert might use it to test the correctness of the system's methods.

How did you come to this question or conclusion? This query is usually interpreted in the context of the reasoning process of the expert system itself. In many rule-based systems, such an inquiry is answered by referring to the assertions and hypotheses and the rule-base itself. Teiresias is an explanation program associated with the MYCIN medical diagnosis system (Barr and Feigenbaum 1982). In the following example, MYCIN asks the user a question, and the user asks why (that is, "Why do you ask this question?").

Then Teiresias takes over, expanding the question and answering it. An expansion is important, allowing the user to be sure exactly what question the system is responding to.

What is the suspected portal of entry of organism-1 into this sterile site?
**** WHY**
[I.e., why is it important to determine the suspected portal of entry of organism-1 into this sterile site?]
[3.0] . . . in order to find out about an organism or class of organisms for which the therapy should cover.
It has already been established that
[3.1] The site of the culture is blood
[3.2] The stain of the organism is gram positive
Therefore if:
[3.3] The portal of entry of the organism is G.I.
[3.4] [Abdomen is the locus of the infection, or Pelvis is the locus of the infection]
Then
There is strongly suggestive evidence (.9) that Enterobacteriacaea is the class of organisms for which therapy should cover.
[Rule 095]

Explanations of the Teiresias sort are not easy to achieve. Teiresias is itself a rather complex expert system. It reasons about the rule base and reasoning processes of MYCIN to achieve its responses. Typically, a shallow knowledge diagnostic system does not contain this class of user assistance. The more expert the user is, the more detailed the assistance that is required. Physicians being what they are, a medical diagnosis system for physicians could not succeed—probably couldn't even get a fair hearing—without being able to explain itself in detail. Diagnosticians of electronic systems, accustomed to using electronic aids, are likely to accept canned expertise. The users of the medical systems, highly trained, are likely to understand medical reasoning at a fairly deep level. (Much more is at stake in medical diagnosis, and the cost of error is much much higher too.) The technician is more likely to accept a phenomenological approach, but that does not let a nonmedical diagnostic system designer totally off the hook. Although the end user might be satisfied relatively easily, the knowledge engineer should not be. As the scope and complexity of the diagnostic knowledge base grows, the designer needs to be able to examine conclusions and deductions at the reasoning level.

By what steps did we get here? It is typical of an accounting program to create an *audit trail,* a history of the operations that would permit an auditor to recreate every interaction. An expert system should be able to do the same. It should be able to show all the questions, answers, deductions. With suitable instrumentation, this display could be structured to show, perhaps in tree form, all the avenues explored. Nothing restricts the usefulness of such a display to the end of the interaction. In a complex process, it would be useful at any point, not only to see the prior steps but to revisit any earlier point, either to explore an apparent dead end more thoroughly or, perhaps, to set off on an entirely different tack.

What do I do next? This question was referred to earlier as the "who's in charge around here" phenomenon. In books on human factors, it is more formally referred to as the balance of control between automation and human control. Shneiderman (1987) presents a lengthy table drawn from several sources listing where humans are generally better and where machines excel. For example, humans tend to be better at recognizing constant patterns in varying situations, machines at sensing stimuli outside a human's range. In an expert system, the abilities and needs of three players must be balanced: the user, the computer, and the domain experts who supplied the knowledge base. To summarize a topic worth an article on its own:

The user should do those operations humans are best at, the computer those operations computers are best at.

If the expert system is an assistant, the user should control the interaction but be able to ask the computer for advice on how to proceed.

If the expert system is designed to be autonomous, the user should expect to be told (better, asked) what to do.

If the user is an expert, the best model is to have the expert system in the assistant role.

If the user is less expert than the knowledge suppliers, the control balance decision should be influenced by the domain experts' judgment on the best mode of operation.

When in doubt, give the computer the assistant's role but supply plenty of what do I do next information.

What do you know about? Do you know about X? Can I do Y? Computer programs do not (yet?) have self-awareness. Nevertheless, although no systems have such a capability today, users of expert systems need to ask such questions; eventually, a program will

A system is a system is a system. No reason exists for an expert system to be any different .

Semantics	**Main dialog for Q/A interaction, how/why options**
Syntax	**Window, text for output, mouse-menu selection for input**
Style	**Window in upper right, always present during interaction; label in window header is left justified, meaningful to the user (in the context of the inter action); the window may not be overlaid by anoth er window; output text, ragged right; horizontal mouse-menu for primary input; visually distinct (optional) how-why menu; CURSOR NORMALLY HERE**
Semantics	**Structural model display**
Syntax	**Physically model device, display status**
Style	**Window on left half of screen, overlayable, nested window of windows, and so on**
Semantics	**History log, output only**
Syntax	**Scrolling window**
Style	**Window, lower right; overlayable**
Semantics	**How (optional), graphic adjunct (optional)**
Syntax	**. . .**
Style	**. . . CURSOR APPEARS HERE WITH MESSAGE "click to exit"**
Semantics	**Graphic**
Syntax	**. . .**
Style	**Window to left of How box; appears with How box, disappears with it**

Figure 6. Informal Abstract Representation of Figure 3.

have to be able to describe its own limits. No guidelines are proposed here except that such questions are almost always in the mind of the user, and the more facile a system is, the more the user expects it to know. If the system cannot recognize a situation outside its limits, the user is not well served. A classic example follows:

User: Did Dick pass home economics?

Computer: No.

The user now concludes that Dick failed home economics. Actually, Dick never took the course. The computer was really answering the question, Is Dick's name on the list of those passing home economics. (I am on the verge of never being able to use this example again. A recent paper described a significant step toward solving this null value problem. [Kao, Cerone, and Luk 1988].)

The principle of parsimony: If the user asks a simple question, the user expects a simple answer. If the full answer is long or complex, it is better to supply a summary and provide the ability for the user (with a single keystroke or mouse click) to request more.

The principle of extrication: The user should be instantly able (with a single keystroke or mouse click) to terminate the assistance request and return to the point where the request was made. The screen or display should be restored exactly to the state before the request except, perhaps, for a record of the request in the history log.

The golden rule: The user often needs assistance but does not know it. A truly expert expert system should recognize and respond accordingly. Research is needed here. "He who sees a need and waits to be asked for help is as unkind as if he had refused it" (Dante, *Purgatorio*).

An Object-Function Model of the User Interface

A reasonable way to go about designing a computer program (once the specifications and goals are defined) is to (1) decide what objects are to be manipulated and what operations can be performed on them, (2) select a representation of the objects and the functions that implement these operations, and (3) compose the data structures from the representations and the program from the functions.

The first two steps might be likened to defining the semantics and syntax of a language. The objects and operations correspond to semantics, and the representations and functions correspond to syntax. In principle, it is well to define the former before the latter. In practice, implementation considerations can require trade-offs and the rethinking of earlier definitions. However, experience has shown that if syntactic considerations can be deferred, it is easier to achieve correctness.

This model can be mapped onto the design of user interfaces. In the semantics part, the objects are the messages, the types of information to be displayed or entered—questions and answers, for example. The operations are what is to be done with these objects. Questions are asked and answers given, for example. By the way, parallel processing is indicated even at this abstract level. When a question is asked, and how or why information is available, appeal is made in parallel to two aspects of the user's attention. The syntax of the interface is, of course, the

implementation onto the screen, printer, mouse, keyboard, loudspeaker, and so on.

It is difficult to characterize the usability and aesthetic aspects of the interface neatly. In the programming languages area, such aspects are depreciated as syntactic sugar, implying they are in some sense unnecessary. It is almost an abstract virtue to avoid them. With the user interface, these implementation details are the most important aspect. They can be considered the style of the interface. Abstracting the style, the mode of implementation, from the syntax provides a useful third part of the interface abstraction. For each class or stream of information to be transferred, its semantics, syntax, and style must be defined. (If semantics and syntax are analogous to strategy and tactics, then style might be an analog of logistics.) Figure 6 shows an informal specification of semantics, syntax, and style of the screen illustrated in figure 3.

Clearly, some of the style options apply to more than one message class and more than one screen. Because style definitions can be global and nested just as other parts of system definition, the following higher-level definition might have been made: All nongraphic windows have meaningful one-line labels left-justified in the window header. Such generalizations are useful. Why, for example, exclude graphics windows? (The answer for CATS is the familiar, "It just happened this way.") With style as with syntax and semantics, it is easier to consider such global issues as consistency at an abstract label. Abstraction also aids the identification of common items. Properly structured, a style definition could drive an interface compiler, but discussion of such a compiler is beyond the scope of this article.

Actually, figure 6 is more like the output of a representation generator. The input is more like building a sample screen with a menu selection plus the drawing package generator tool currently under development. The work is considered proprietary, but it bears some resemblance to the methods described in a recent article by a friend and former colleague, Anatol Holt (1988).

The User's Conceptual Model of the System

Earlier, the user's understanding of, and expectations for, the system were discussed. These are just part of the model of the system the user has internalized. Despite any formal documentation, training, and experience, interacting with a complex system is a form of gedanken experiment. The user forms hypotheses about the system and works within the bounds of the hypotheses. When experience shows different behavior, one of two events occurs. Either users ignore, rationalize, or just don't notice the inconsistency, or they adjust the model, perhaps by explicit experimentation.

In the former case, it is difficult to predict what can occur. It is possible that the user model will significantly diverge from the true behavior. (Incidentally, the definition of true is subjective. What users see is the truth to them. If a system is hard to learn, users strongly resist changing their model even in light of objective evidence.) Humans tend to forget that computers are deterministic. A common response to the unexpected behavior of software is to pretend it didn't happen in the hope that it won't happen again. The flaw in the original Space Shuttle control program that delayed the first launch was observed twice during practice runs by technicians who responded by restarting the system. The actual bug had a 1 in 148 chance of appearing at the power-on stage. In both cases, because restart fixed the problem, no one followed up. The day of the first launch was one of those 1 in 148 instances, and the launch was aborted.

Such human behavior must be taken into account in several ways in the design of expert system interfaces. In the first place, the designer must assume that the system and its documentation are not fully correct or complete and that the user will not fully understand what the system does. Assume the user will make errors. Then design the interface to help keep the actions of the system clear to the user. The following might improve the chances of a correct user's model:

First, provide a focus of attention. (In CATS, the cursor was enlarged from the normal small arrow. In general, the user's attention is directed to the point [or at least the window] where the cursor is. See, however, the discussion on changes of state.)

Second, at all times, make clear the full range of operations available to the user at this point in the interaction. (If How and Why buttons are not present, the user's model might omit them at critical times.)

Third, omit unnecessary items from the display. (If the system model is not relevant to a given interaction, remove it from the screen. If it stays there when not needed, the user can begin to ignore it.)

Fourth, make changes of state obvious. (A hard one: When the state of a device in the CATS system model changes, the new value flashes a few times in an attempt to bring it

The expert system, especially if capable of reasoning and inference, is working by methods not likely to be understood by the user.

to the user's attention. Actually, the user's attention is usually fixed elsewhere, and the flashing is ignored or, at best, stopped by the time the user notices. Color, if available, would be useful here.)

Making the system friendly will help motivate the user to spend enough time learning the system to build a correct model. The following guidelines are recommended:

First, encourage explicit "what would happen if" experimentation by making it easy for the user to checkpoint the state, experiment, and return to that state.

Second, log errors, annotate the log to make the context clear, and make the log available to the user. (See next item however.)

Third, ensure the privacy of an interaction. (Experimentation is more likely if users feel their errors will not be seen by others.)

Fourth, don't be pushy. (If a response to a question is required, wait a suitable time and then offer assistance. If it's the user's turn to take an action, wait a longer time and then indicate possible alternative actions. In neither case, force response to the assistance offer. The user might want to work on the original item.)

The System's Conceptual Model of the User

It is reasonably well accepted that "user models are an essential component of any system that attempts to be 'user friendly', and that expert systems should tailor explanations to their users, be they super-experts or novices" (Sleeman 1985). Although some studies limit their attention to user-requested assistance in expert systems, others treat more general areas such as tutoring systems and knowledge acquisition (Boose 1985; Zissos and Witten 1985). The extent to which an entire system needs to be specialized depends—as always—on the types of users and their expertise. It is relatively easy to accommodate a single class, but it is not known how to handle all at once. Based on extrapolations of recent research, systems will first be made to cope with users by class and then with users as individuals. In effect, the system will have to build and use a database keyed to individuals. In slightly fantastic terms, the system will have to know its users, learn their needs, and plan its actions accordingly. The words know, learn, and plan were deliberately chosen to identify the research that is needed to build tomorrow's systems.

Models come in two flavors, canonical models that categorize an abstract user (perhaps tied to the level of expertise) and individual models which are identified with individual users. If a long-term relationship exists between the user and the system, the latter sort is possible. Today, cognitive modeling at the level of the individual must be considered as a sort of Holy Grail. It might never be found, but there is virtue in the search. The following points summarize some of the work done and some that is still required.

Knowledge about users (by experience class): The system of the future will be designed to behave differently for different levels of user. Users will be categorized by level in terms of expected knowledge and expected behavior. For example, an expert might be asked, "Is the power supply functional?" The novice might get the question, "Are there any warning lights on the power supply?" If an inexperienced user hesitates a long time before answering, it would be well for the program to take the initiative—offer advice or make a suggestion. With an experienced user, unless the wait gets excessive, better to let well enough alone. Conceivably, certain sorts of group behavior must also be accommodated. With a lone user, it is possible to draw conclusions about the reason for a delayed answer. If two people are working together, unless it is certain that both are rank beginners, it would be best for the program not to take any initiative.

Knowledge about an individual user: How might the machine tell a user's level of expertise at first? It is best to just ask and take the answer with a grain of salt.

Adaptation and learning: The system will have to be able to adapt to changing patterns of behavior. In keeping profiles of individual users, it will be possible to learn an individual's behavior patterns; note changes; and by using the stored taxonomy of models, change the interface to meet a new set of needs. It is unlikely that any AI system will be able to learn totally new behaviors by observing users. However, it is possible that with active human aid, the taxonomy could be periodically extended. If the machine is permitted to store interaction profiles, existing classification schemes could be used to seek patterns that could be exploited, for example, in understanding the reasons for repeated errors.

Reasoning and planning: It is quite clear how user modeling on a class or individual basis could be exploited in user assistance and tutoring systems. How to make more general use of such an ability in general is not clear. Nevertheless, even if limited to these two areas, individualized behavior would be worth the research effort. One way to achieve the desired behavior is to apply planning methods. Work has been done toward model-

ing the user in a teaching assistant by inferring the student's problem-solving plan and using this plan to generate advice to the student (Genesereth 1982). In a diagnosis system, the overall plan of the user is clear: Find faults and fix them. In a system designed as an expert's assistant, this methodology could be applied to building a hierarchy of plans about the user's goals and subgoals, tied to the symptoms and observations. A base of such plans could then be used to drive the system for a less experienced user.

Is it really necessary for an expert system such as the CATS Tomoscan application to have so facile and so complicated a behavior? No, but the current state of the art in expert systems is only the shadow of things to come. Consider only the limited domain of diagnosis and repair of medical equipment. Today, when a Tomoscan occasionally fails, a technician arrives and fixes it. Future generations of diagnostic equipment will be orders of magnitude more complex—as will their maintenance. The first line of defense will be self-diagnosis, but it is unlikely that an electronic-optical-mechanical system of this magnitude will ever be fully self-testing. Users (who will probably be interacting with an expert system during normal operation) will handle triage and first aid with the aid of a diagnostic system running on the machine itself, and this user might be anyone from a repair technician to a medical technician to a physician. Here is where the adaptive, learning interface will be needed.

Is there any hope of quick progress? In the words of an active researcher, "Conceptual modeling is all the rage these days. The problem is domain knowledge. That is, there's no good way to model a 'generic' user. The best systems use pre-knowledge of the domain the user is working in and make some assumptions about his level of expertise" (Wexelblat 1987). That is, in terms of responding to a user's stated level, probably yes. In terms of a true individual user model, probably no. In summary, significant improvement is likely. Truly intelligent interfaces, however, remain in the misty future.

A Warning and a Conclusion

I have tried to clarify some issues concerning the interaction between humans and computers in the realm of expert systems and, I believe, to other sorts of AI programs. In doing so, it has been necessary to use the euphemisms and anthropomorphisms of the computer community. Such uses often mislead those outside the community. Working within the field, we must not become inured to the dangers of misleading others and ourselves(!).

> I don't quite know whether it is computer science or its subdiscipline Artificial Intelligence that has such an enormous affection for euphemism. We speak so spectacularly and so readily of computer systems that understand, that see, decide, make judgments, and so on, without ourselves recognizing our own superficiality and immeasurable naivete with respect to these concepts. And, in the process of so speaking, we anesthetize our ability to evaluate the quality of our work and, what is more important, to identify and become conscious of its end use (Weizenbaum 1986, p. iv).

When we say a computer calculates, we mean that it is doing a process analogous to human calculation resulting in the same answer. Whether that process is truly analogous to the human mental process of calculation is irrelevant. Mechanical methods exist that can tell if the computer is computing correctly. When we say that the computer is an expert, we are treading on ground which is less firm. It is the responsibility of the knowledge engineer to make sure that the knowledge is complete, consistent, and correct and that the inference methods draw the correct conclusions. We will eventually have to learn to trust these systems, or we might as well abandon the effort to build them, which is what explanation systems are all about. The responsibility of the system designer is to include as much intelligence or pseudointelligence as possible. The responsibility of the designer of the user interface is to make the workings of the system as clear and apparent as possible and to make the use of the system as simple as possible.

> Simplicity, simplicity, simplicity. I say, let your affairs be as two or three, not a hundred or a thousand; instead of a million, count half a dozen, and keep your accounts on your thumb nail (Thoreau, *Walden*).

The boundary between science and art (mysticism?) is sometimes obscure. Expositions of the human-machine interface, although claiming to be the former, tend toward the latter. Precious few hard data are supported by rigorous statistics in the field. Those which have been verified tend to be microproblems (what is an optimal number of items in a menu?) rather than global issues (automation versus human control). Evidence tends to be anecdotal (well, we did it this way, and no one complained). Controlled experi-

ments are hard to plan, hard to do, and hard to find in the literature.

Many of the principles described here are included in the Tomoscan diagnostic system. A cognitive psychologist, Dr. Maddy Brouwer-Janse, was involved in the early 1988 field trial, and her paper describing this experience might be published later. In the meantime, system designers are urged to give the ideas espoused here a chance. If you agree with them, try them out. If you disagree and have better ideas, I'd appreciate getting copies of your write-ups. Alternatively, please consider contributing to a revised edition of this article. ■

Acknowledgments

The work described in this article was performed while the author was at the AI Research Department at Philips Laboratories, Briarcliff Manor, NY 10510.

The initial Tomoscan diagnostic prototype was implemented by Dr. David Schaffer with the assistance of Mr. John Martin. The CATS implementation was done by Dr. K. P. Lee, Dr. Paul E. Rutter, and Mr. John Martin. Dr. Rutter had responsibility for the design of the user interface. Dr. Jorge Caviedes, Mr. Tejwansh Anand, and Mr. Michael Reed are the current members of the knowledge engineering group supporting CATS. Mr. Robert Purdy of Philips Medical Systems in the United States and Mr. Derek Fosberry of Philips Medical in Holland were the (patient) domain experts whose experience was mined for the knowledge base.

I thank those who commented on drafts of this article: Lee, Rutter, Schaffer, and unnamed reviewers. Thanks also to Alan D. Wexelblat of Texas Instruments who pointed me to a significant part of the recent literature on conceptual modeling.

References

Barr, A., and Feigenbaum, E. A. 1982. TEIRESIAS. In *The Handbook of Artificial Intelligence,* Volume 2, eds. A. Barr and E. A. Feigenbaum, 87–101. Los Altos, Calif.: William Kaufmann.

Boose, J. H. 1985. A Knowledge Acquisition Program for Expert Systems Based on Personal Construct Psychology. *International Journal of Man-Machine Studies* 25:495–525.

Carroll, J. M., and McKendree, J. 1987. Interface Design Issues for Advice-Giving Expert Systems. *Communications of the ACM* 30(1): 14–31.

Genesereth, M. L. 1982. The Role of Plans in Intelligent Teaching Systems. In *Intelligent Tutoring Systems,* eds. D. Sleeman and J. S. Brown. Orlando, Fla.: Academic.

Holt, A. 1988. Diplans: A New Language for the Study and Implementation of Coordination. *ACM Transactions on Office Information Systems* 6(2): 109–125.

Kao, M. N.; Cerone, N.; and Luk, W. S. 1988. Providing Quality Responses with Natural Language Interfaces: The Null Value Problem. *IEEE Transactions on Software Engineering* SE-14(7): 959–984.

Lee, K. P.; Martin, J. C.; Rutter, P.; and Wexelblat, R. L. 1988. A Development Environment for Diagnosis Tools. In Proceedings of the IEEE Conference on AI Applications, 262–267. Piscataway, N.J.: IEEE Computer Society Press.

Schneider, M. L.; Wexelblat, R. L.; and Jende, M. 1979. Designing Control Languages from the User's Perspective. In *Command Language Directions,* ed. D. Beech. Amsterdam: North Holland.

Shneiderman, B. 1987. *Designing the Human Interface.* New York: Addison-Wesley.

Shneiderman, B. 1979. *Software Psychology.* Cambridge, Mass.: Winthrop.

Sleeman, D. 1985. UMFE: A User Modeling Front-End Subsystem. *International Journal of Man-Machine Studies* 23:71–88.

Sleeman, D., and Brown, J. S., eds. 1982. *Intelligent Tutoring Systems.* Orlando, Fla.: Academic.

Weizenbaum, J. 1986. Not without Us. *Computers and Society* 16(2–3): 2–7.

Weizenbaum, J. 1976. *Computer Power and Human Reason.* San Francisco: Freeman.

Wexelblat, A. D. 1987. Personal electronic communication, 8 March.

Wexelblat, R. L. 1983. Designing the Systems of Tomorrow. *Electrical Communication* 58(1): 93–97.

Wexelblat, R. L. 1981. Design of Systems for Interaction between Humans and Computers. Paper presented at the 1981 Meeting of the British Computer Society, London, 15-17 August.

Zissos, A. Y., and Witten, I. H. 1985. User Modeling for a Computer Coach: A Case Study. *International Journal of Man-Machine Studies* 23:729–750.

Notes

1. Occam's Razor: Entities should not be multiplied unnecessarily.

Richard L. Wexelblat recently joined the Institute for Defense Analyses as the deputy director of the Computer and Software Engineering Division. Prior to this position, he was research department head for artificial intelligence at Philips Laboratories, where his group was active in research and in developing practical applications of AI methods. Wexelblat received a bachelor's (1959) and a master (1961) of science in electrical engineering from the Moore School of Electrical Engineering and a Ph.D. (1965) in computer science from the Graduate School of Arts and Sciences at the University of Pennsylvania. He is a member of the American Association of Artificial Intelligence, American Association for the Advancement of Science, Association for Computing Machinery, and Institute of Electronics and Electrical Engineers.