Ease of use: A system design challenge

by L. M. Branscomb
J. C. Thomas

While it is becoming increasingly obvious that the fundamental architecture of a system has a profound influence on the quality of its human factors, the vast majority of human factors studies concern the surface of hardware (keyboards, screens) or the very surface of the software (command names, menu formats). In this paper, we discuss human factors and system architecture. We offer best-guess guidelines for what a system should be like and how it should be developed. In addition, we suggest ways in which advances in research and education could result in systems with better human factors. This paper is based on an address by L. M. Branscomb and a publication by the authors in the Proceedings of the IFIP 9th World Computer Congress, Paris, France, September 19–23, 1983.

As the end users of computer equipment continue to diversify, the first claim on the greater computing power that improved microelectronics can provide must be that of making computers easier to learn and easier to use. It takes a smart machine to serve a naive user well. Fortunately, technological progress is enabling programmers to do things that previously would have been too costly.

The personal computer, for example, has totally changed the end user's perception of what constitutes an acceptable user interface and system behavior. Integrated systems combining word processing, financial analysis, graphics and data management functions, and windows that display different tasks simultaneously, are typical features of many ingenious programs being written to help any motivated, intelligent person progress rapidly in accomplishing useful tasks.

Despite the current fascination with stand-alone personal computers, however, the future undoubtedly lies closer to *cooperative processing* between intelligent workstations and central-site computers. To exploit the synergism that can result from taking both these machines and properly allocating tasks between them, the burgeoning research in human factors issues must broaden its scope.

Though a significant number of questions remain in the area of hardware human factors, much is known. A body of knowledge also exists concerning such software issues as the structure of command languages, menus, and error messages. A perusal of the literature[1–3] confirms this focus, for the most part, on the important but surface aspects of the user interface.

Yet, anyone who has attempted to improve the human factors of a computer system by altering the surface structure after all the fundamental architectural decisions have been made feels continuing frustration. We know that the system architecture has significant and widespread implications for user-friendliness, but we know very little about how to make fundamental architectural decisions differently as a result of an emphasis on human factors. Extensive study is needed to collect data for such decisions.

The purpose of this paper is to begin laying the groundwork for that study by focusing attention on relevant issues in four areas. First, we provide our current best guesses about what a system architecture should be like and offer guidelines for the system

designer. Second, we suggest how a new set of software development tools could not only increase the productivity of the software development process, but also increase the likelihood of developing a system with good human factors. Third, we briefly point out some areas of fundamental and applied research that are needed before we can more adequately articulate the impact of architecture on ease of use. Fourth, we make some suggestions for changes in educational policy.

Throughout this discussion, we attempt to go beyond the usual description of the man-machine interface to look at some of the more fundamental questions about how computer systems ought to be conceived, in the first instance, so that they will be easier to use. Our concern here is not so much *artificial intelligence*—the task of building a computer to reason as we imagine people do—but what might be called *artificial personality*—how to make the computer behave the way we want it to.

What a system architecture should be like

People communicate via, not with, computer systems. Alphonse Chapanis, at the Johns Hopkins University, was perhaps the first to point out that the phrase "man-machine communications" is misleading. Instead, as he and others have suggested, it may help to think of a terminal as a channel through which a user communicates with other human beings—those who designed and programmed the system and put data in its memory. Some computer scientists, especially those attempting to build systems that exhibit artificial intelligence, find that view limiting.

Certainly, in the engineering sense, people do communicate with, as well as via, computers. But we believe it is counterproductive for systems designers to overlook obvious differences between human beings communicating via a computer and information transmission between two computers—between computer "thinking" and human thinking; between computer "goals" and human goals.

There need be nothing ontologically more mysterious about helping people by incorporating relevant information in computer systems than there is about the process whereby experts write textbooks. Consider John Seeley Brown's[4] series of tests that quickly determine what wrong algorithm a child is attempting to use to solve math problems. Children taking these tests are, in a real sense, communicating with the author of the tests. Similar diagnostic system responses to untrained users of computers could be a powerful help to people who want to learn by trying, or are confused by the machine's responses to their actions.

The computer allows us to apply human communications expertise to the human communication process.

Enhancing the human communication process. Electronics allows us to communicate with people over long distances. Yet, as currently constituted, electronic communication takes place over a much narrower bandwidth than face-to-face communication allows. In addition, the communications system itself often provides a less than obvious set of commands, messages, protocols, and so on. Human factors research and experience are beginning to reduce the unnecessary glitches to communications, and further reductions in the cost of technology will allow wider bandwidths as well.

The power of the computer, however, offers the opportunity to do far more than simply reduce the effective distance between people. The computer allows us to apply human communications expertise to the human communication process.[5] Even the most casual observation reveals that humans often have difficulty in face-to-face communication. Some of these difficulties can be traced back to human cognitive difficulties such as keeping track of multiple goal contexts, holding to cognitive complexity, and clearly separating statements of feeling from statements about the real world.

In today's world, people are typically pursuing multiple goals in various contexts, and a computer could help them deal with this complexity by providing goal reminders. A computer filter could search for terms that are likely to be oversimplifications in social contexts, such as "never," "always," and "should," and could request clarification. A system

could encourage a separate channel for the expression of feelings if that is agreed upon as appropriate. These aids might help people communicate more effectively. The point for architecture is that a system for person-to-person communication might well allow the possibility of a third-party human arbitrator to help the communication process. Analogously, there might be access to code that performs some of an arbitrator's functions.

The computer may be partly responsible for the information explosion, but it can offer solutions.

It may also be important for an electronic mail system to lend itself to highly interactive "chit-chat," before "serious" information exchange commences. Tests have shown[6] that the most rapid form of communication when exchange of acquired knowledge is attempted is in short, fragmented phrases of a few words—highly redundant and highly interactive. Also, a computer communication system may seem threatening if the users are deprived of the occasion to socialize, to establish the needed relationship.[7]

If we take as a premise the widespread existence of computer networks, a number of interesting further possibilities arise. The computer may be partly responsible for the information explosion, but it can also offer potential solutions. For example, it may be possible—given a graphics, video, and text terminal with synthetic speech output—for one human being to communicate thoughts much more quickly and completely to another human being. Evidence already suggests that rapidly presenting words and pictures in the same place may allow faster reading than ordinary scanning.

New ways of doing work. Once we postulate the existence of such networks, we can imagine that people will evolve new kinds of organizations for doing work that take better account of the individual knowledge that people possess, as well as differences in abilities and preferences. For example, in solving a problem requiring a creative solution, a group of people may independently generate ideas. These people could be chosen for their divergent thinking ability and, over large networks, they could be chosen from diverse cultural backgrounds. Indeed, they may be largely self-selected.

By asking for suggestions independently, the system could avoid premature clashes of opinion that might occur with people of such diverse backgrounds. The computer system could then automatically form the union of these suggestions and distribute it for another round of ideas.

Once a set of ideas were generated, a different set of people—ones particularly good at judgment—could be polled independently for their evaluations. Such a process would not be appropriate for every case that requires creative thinking, but it illustrates just one of the ways[8] that computer networking could allow different divisions of thought, labor, and communication patterns than could exist in face-to-face groups.

In nearly every large organization, there are informal groups of people with common interests, and there are also formal committees to solve problems not covered by the hierarchical structure. Allowing such groups to communicate via computer networks focuses the composition and work product of such groups on those persons with relevant interests and expertise. Communication via computer networks also avoids irrelevant factors such as geographical location or even preference for certain accents or physical appearances that merely lowers the probability of success. This is already happening in organizations with large open-ended, peer-connected networks, such as the IBM worldwide VNET, which contains over 1000 host processors in over 125 cities in some thirty countries.[9]

To the extent that networks—no less than single-user systems—exhibit good human factors, they will be used by people with relevant expertise. If network structures constrain communication paths to the official organization chart of the enterprise, the result will be to decrease the effectiveness of the individuals in the enterprise. Organizations do not function in the way authority charts suggest.

Architectural guidelines

To the extent that arbitrary and unwieldy conventions exist, the use of computer facilities will be gated by expertise in knowing these conventions. It may at first seem as though mandating a strict set of standards for user interfaces should be done quickly. Unfortunately, however, we do not have enough knowledge about software human factors to do this intelligently. The following sections do not present proposals for standards but rather best-guess guidelines that are to be thought of as tools of thought for designers and issues for research in human factors.

Separate the user interface. Perhaps the most important architectural guideline, given our current state of ignorance, is to separate as cleanly as possible the user interface from the rest of the system so that future improvements can be made to the user interface based on empirical data without requiring a complete rewrite.

Without changing the logic of the system, messages should be translatable into different languages.

Imagine, for example, that one is designing a system for helping office principals define problems. To the extent possible, one should define the interface as a set of states and transitions. Associated with each state is a set of fundamental choices the user could make, each of which will take the user to a new state. Associated with the user's choice is a pointer to a list of feedback messages, and associated with the state is a state-message list. Associated with each invalid response in a given state is an error-message list.

Without changing the fundamental logic of the system, the messages should be translatable into different languages. Different interfaces should be amenable to experimentation by simply specifying the number of the item in the list that will be displayed.

Figure 1 Simplified Audio Distribution System transition table

SIMPLIFIED ADS STATE TRANSITION TABLE

USER RESPONSE (KEY NO.)	ACTION (ROUTINE CALLED)	STATE TO WHICH CONTROL TRANSFERS	SYSTEM MESSAGE NO.	COMMENT
2	COSLINE	TCUST	12	CUSTOMIZE
3	NONE	TDISC	0	DISCONNECT
4	COSLINE	XGET	0	GET
5	COSLINE	XLIST	0	LISTEN
6	NONE	EMPTY	6	UNDEFINED
7	COSLINE	XRECD	0	RECORD
8	COSLINE	XXMIT	0	TRANSMIT

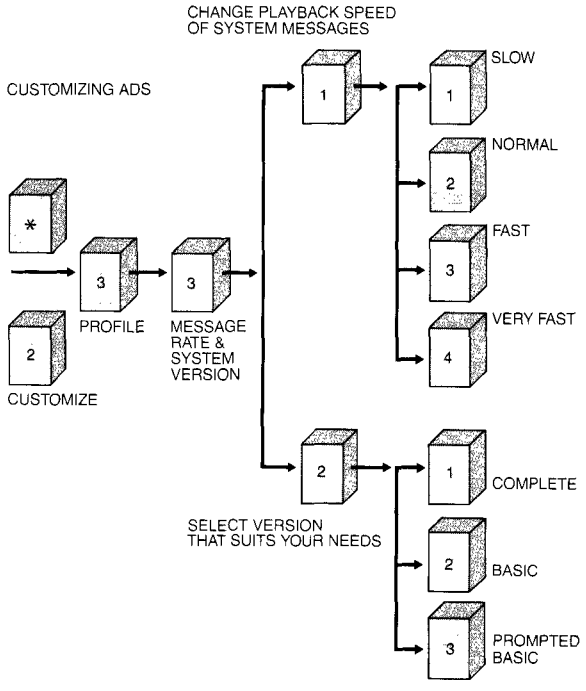
Error message generator routines should have access to the state as well as the user's response. If empirical observation later suggests that a single error message can cover a variety of cases, such as the message "Syntax error in line 20," the messages can be collapsed.

It is also possible that experience with prototypes will reveal that certain user functions are really not useful. This should translate into certain states that are rarely visited. In such cases, it may be decided to delete such states. With such a scheme, the interface state table itself should provide useful information to allow the programmer to delete such states easily while minimizing unnoticed side-effects. To a great extent, the IBM Audio Distribution System follows these principles.[10,11] The interface is defined as a set of states and transitions. (See Figure 1.) Associated with each state is a set of fundamental choices the user can make, each of which takes him to a new state. Associated with the user's choice is a pointer to a feedback-message list, and associated with the state is a state-message list.

During development, the system was amenable to experimentation simply by changing the numbers of the items on these lists. Now, without changing the fundamental logic, messages are easily translatable into different languages. In addition, error messages that are discovered to be confusing can easily be changed.

Layered interface. Various users of nearly any system have different degrees of sophistication and experience, and the interface could reflect these differences. For example, suppose that a menu-driven interface may be necessary for the new or casual user, but that

Figure 2 Customizing the Audio Distribution System



a command-driven interface is better for a person who uses a system very often. Using the scheme outlined earlier in this paper, one can have a flag in each user's profile that indicates which interface is appropriate to that user. This indicator can be changed either by user choice or by an algorithm in the system based on the number of interactions between user and system.

In addition to having an interface with both commands and menus, there are other ways in which the system can be different for different users. One can also implement a layered interface by having several groups of commands starting with a small group of useful and fairly intuitive commands. Initially, the user need only be told about these. Later, the system can explain further branches.

Some of the ideas outlined here are also implemented in the Audio Distribution System. In this system there are both command interfaces and menu-driven interfaces. Users can specify which interface they prefer. The system messages, which in this case are audio, can be played out at different rates depending upon the user's experience. (See Figure 2.) Additional research along these lines has shown empirically the value of restricting the options available for new users.[12]

Media translatability. In office systems for principals, there are individual differences in the preferred type of media input and output. Some office principals are quite willing to type and can do so proficiently; others insist on handwritten input or dictating. The details of the user interface must be somewhat different for these input media.

Different output media also have implications for the user interface. For instance, messages that appear on a visual display unit should probably contain the most informative part of the message first to allow rapid scanning. However, if the information is to be provided by synthetic speech, it is probably better to begin the message with a more predictable preface that gives the listener time to "tune in" to the possible endings of the sentence.

As another example, it is quite reasonable to present twenty logically arranged menu items simultaneously on a visual display. However, an audio menu of more than three or four items becomes impossibly difficult to recall and use.

The application programmer should not have to program in such a way that the code dealing with these devices is intermixed with the code that determines the logical structure of the application. Rather than specifying, for example, where on the screen an item should appear, the application programmer should only have to specify the relative importance of various kinds of information. A separate part of the design should deal with the issue of how to display various kinds of information, given the particular I/O devices currently being used.

Behavioral observation hooks. Given our current state of knowledge, there is no way that a designer can be guaranteed to know how a tool will ultimately be used. Therefore, it becomes extremely important for systems not only to reflect educated guesses about what a system should look like; it also becomes necessary for them to be easily modifiable. Even if the design is exactly right for today's situations, tomorrow will be different, with a slightly different set of requirements. Therefore, it is very important for the system to be adaptable.

If the system is going to adapt, we must ask, "Adapt to what?" We hope that the system will adapt to the

needs and preferences of the users. If this is to happen, however, there must be some facility for finding out what the users are doing. It would be very useful, therefore, at least in prototypes, and probably in products, to have the mechanisms available for keeping track of general trends in user behavior.

For example, if there is an on-line help system available, it would be useful to know, for each help panel that is called, what happens next. Does the user turn off the machine, use the asked-about command without error, ask for help on another item, or use the command and make an error? Which of the allowable transitions between states does the user really make use of? Are there menu items that are simply never chosen? If so, one might question whether they really need to be there for as-yet-unencountered emergencies or for symmetry. Of course, there is the possibility that unused menu items are simply mistakes. One might question another design consequence. If it turns out that one tenth of the links in a network are used 95 percent of the time and they turn out to be the slowest, one must question whether this is a design accident, and whether the other five percent requires the higher speed.

It is difficult for a user of several systems to recall the specific name of a particular system.

One may buy a cheaper dedicated line for communications, and this may seem like a tremendous savings. But what are the behavioral data? How often must people redial because a cheaper line is in use? How many additional telephone calls are missed? If one has such data, one can begin to calculate whether the added human cost is worth the cheaper line rental. Having facilities in the system to collect behavioral measurements also makes possible better synonymity and adaptability. These properties are discussed next.

Synonymity. In natural language, we can refer to the same thing in many ways. It is very difficult for a casual user or even an expert user of several systems to recall the specific name required in a particular system. Quantitative work[13] has shown that regardless of how well the name for a command or file is chosen, there is a low probability that someone else will spontaneously guess the same name (i.e., approximately 0.15).

These investigations suggest a general synonym table as a means around this problem. When a user enters any one of several names, the effect is the same. Using such a scheme over a wide variety of problems, a correct unique item is picked about 75 percent of the time. In the vast majority of the other cases, a disambiguating menu could appear which would consist typically of two to four items.

Whenever an item is referred to frequently in a way not anticipated by the synonym table, behavioral recording facilities could automatically add that item to the table.

A potential difficulty with this approach is that users seem to assume a one-to-one mapping between names and functions. Breaking this mapping could have the effect of actually confusing the new user. A less drastic proposal might be to include all command synonyms in the indices to manuals and in on-line help systems with pointers to the correct name. Thus users who wanted to "stop" or "end" a session could quickly discover that they were to use the LOGOFF command.

Adaptability. Given that one has the hooks in the system for keeping track of the frequencies and sequences of events, one has not only the data that allow a more intelligent system redesign by a human being, but also the beginnings of what is necessary for a system that adapts itself to the changing needs of its users.

The system as it appears to the user. Although it is difficult to specify in general terms what an architecture should be like internally, one can make some further suggestions about how the interface should appear to the user in fundamental terms.

Each of the possibilities presented in the following four sections must be taken as an other-things-being-equal suggestion. For example, if a system requires fast response time, the impact of allowing the user to back up one level may be too great to be worth the increased ease of correcting errors.

Home base. It is characteristic of human problem-solving behavior, in a wide variety of contexts, that people often start over when given the chance. This is also true where user interfaces allow it. For instance, the CHART Utility on GDDM, the Audio Distribution System, and the IBM Personal Computer all have the capability of starting over in a known state. In each instance, these facilities are often resorted to when a user becomes confused.

Commands and options should form logical, coherent gestalts.

Undo. An undo or back-up function (sometimes called "padded cell") is very useful for two reasons. Users may inadvertently hit a wrong key or otherwise make an error of which they are immediately aware. Or else they may choose a command or menu item and not realize until the indicated action is taken that it is not at all what they had expected on the basis of the command name or the menu item description. For these two situations, which are fairly common, it should be quite easy for the user to back up. Even the fairly intuitive notion of undo turns out, upon closer examination, to need considerable analysis and empirical research.[14]

No garden-pathing. Imagine a system with which you are trying to send a message to someone. The system first asks you for the recipient's ID. It then asks you to type in your message. Then the system asks you to confirm that you really want to send this message. Only after all that does the system tell you that the user specified does not exist on the system (and, by the way, it erases your message). This is leading the user down the garden path.

Although it may be easier for the programmer to do all error checks at the end of some transaction, it is more sensible in most interactive systems—particularly where there is a fairly large probability of user error—to check each user input as it is given.

Commands into structures. Commands and options should form logical, coherent gestalts. Work by several investigators[15] indicates that command functions and the names given them should be reasonably predictable. One example of unpredictability is a system where mail is sent by typing SEND, but the command to send a message is MESSAGE (unless the recipient is on another node, in which case it is RMESSAGE). Some query commands on the same system are even less consistent: either Q T or QT to learn the time, Q FILES (but not QFILES) to query files, and ARCV Q to learn what is in archival storage.

Architectural considerations based on basic human capabilities. In addition to the suggestions just given about how a computer system should be internally and how it should look to the user, a consideration of the basic capacities of human beings offers further suggestions for computer architecture.

Of course, an actual computer system must also be designed with costs in mind. A system that costs so much that no one can buy it is not well designed for human factors. Therefore, the following suggestions as to what the ideal computer system might be like must be tempered with the realization of what is currently practical.

One of the fundamental principles of human information processing which has not been capitalized upon is that people can input and output more information when that information comes in and goes out over more sensory channels. The more separate dimensions there are available—for example, in a mixed audio-visual display—the more information a person can perceive. Similarly, people can communicate more information if they are able to use facial, hand, and body muscles as well as their vocal muscles for output.

Purely from the point of view of maximizing human performance, regardless of system cost, all systems today limit too severely the number of ways that information may be provided to the person and the number of ways that the person can provide information to the system.

A second basic principle that is too often ignored in system design is that people are generally error-prone. Further, to avoid error by slowing down requires successively larger slow-downs for small additional gains in accuracy.

People can be very fast at being approximately accurate and are painfully slow at being exactly accu-

rate. (This explains much of the superiority of a good word processor over a typewriter.) There are, of course, times when one must be very accurate, but theoretically, a user could transmit a larger amount of information to a computer if the system were designed to allow greater speed and less accuracy.

One example of planning for error is in an information retrieval system. If we require the user to specify exactly certain fields of information about a required document to be retrieved, it may take an exceptionally long time for the user to recall all the parameters accurately. Suppose we allow an almost-equal type of facility, in which the user can specify what he recalls about a document immediately. Then the system retrieves further information about possible candidate documents and displays them. Thus the person can quickly home in on the exact document desired. Other examples are the concepts of the word processor and the erasing typewriter.

Development process

Apart from our suggestions of what the architecture of a system should be, we also suggest that the development process has a great influence on the quality of the human factors of the resulting system. The way to encourage architects of a computer system (and the implementers, documenters, and testers) to make reasonable decisions is to embed into the development process a set of tools that meet the following criteria:

- Make the developer's job easier.
- Ensure consistency of viewpoint, definitions, and conventions across developers.
- Lead the developer implicitly to use any best-guess guidelines unless they are explicitly overridden.
- Provide for temporal integration of the development process from architecting to implementing, to documenting, to simulating or prototyping, to end-user testing, to field testing.

Design tools. What should such design tools look like? First, they would lead the designer through a design process that would begin by forcing the designer to consider questions that are key to ease of use. For example, the tools may begin by asking the designer to define the classes of users that the system will ultimately serve, the users' tasks, specifications of tasks that have been analyzed in detail, and additional studies required.

Let us suppose that one type of task we are interested in is that of supporting office principals in storing and retrieving documents. Designers could be asked

We should ask the designer the user's goals for document storage and retrieval.

a structured series of questions that prompt them to specify their ideas and data about how office principals do, in fact, now store and retrieve documents; how they can do this best, given their goals and what we know of human capabilities; and how the system intends to move users from their current mode of doing things to the theoretical optimal mode.

Going deeper into the decision hierarchy, we would ask the designer what the user's goals for a document storage and retrieval process are, and how the designer knows this. The designer could proceed without any empirical basis for saying what the user's goals are, but the system should force him to be aware that that is what he is doing. If the designer decides that it would be better to know the user's goals but is unable to provide any empirical basis for stating them, the design system should be able to suggest methods of collecting such data and to retrieve relevant background data that already exist.

A designer of a retrieval system, for example, should be reminded by the system that for a literary reference, people do not generally remember the exact date and often do not know the correct spelling of the author's name, but they are likely to have partial knowledge of each one. This tells the designer that a user should be able to provide some data about a number of descriptors of a desired document so that the system can do a best fit to the incomplete criteria from among the stored documents.

This consideration suggests a number of interesting problems in system design. Perhaps, if fast retrieval is vital, a large storage capability is desirable, in which documents are held in secondary storage, while primary storage contains a set of parameters

and a verbal description for each document. Then a preliminary match for the "nearest neighbors" to the document as specified by the user can retrieve the textual descriptions of those documents (including the author's name, exact title, date, topic, etc.) for menu selection by the user.

Meanwhile, all these near-neighbor documents can be moving from secondary to primary storage so that once the user selects one, it can quickly be displayed. If the choice of one of these documents is confirmed, related documents (those by the same author or on the same topic) can begin moving from secondary to primary storage, based on the high probability that they will be asked for next.

The point is not that such a system design decision is the right one. Rather, it is that the system design tools should force the designer to think about such issues, examine whether empirical data are available, encourage the collection of data, provide a number of design suggestions, and then allow the designer to make a choice. Such a system design tool requires only current technology. The tool need not make any design decisions, but merely serve as a structured reminder and organizer.

Evolutionary systems. While much can be done to induce the designer to pay attention to human factors, it can be argued that in a very real sense we have not yet reached the point in the evolution of computer systems where complex systems are designed from scratch. A more accurate characterization would be that they *evolve*, which seems to be a term that J. Christopher Jones[16] would apply. This is not necessarily a bad thing, but we should be aware of the world as it is at the same time we try to make it a more rational one.

In some cases, evolutionary systems have proved quite effective, e.g., the IBM internal VNET network mentioned earlier. VNET began in a "bottom-up" way when two laboratories working on a joint project needed to exchange data. Soon other related sites were added, until nearly all IBM scientific and engineering locations worldwide are now part of it.

No one designed VNET from the beginning. Various users of the system designed various facilities and pieces. Some facilities are used, and they flourish. In other cases, someone built a function, offered it to the community, and no one used it. Many problems associated with designing for ease of use are avoided when the users build the system. On the other hand, this evolutionary approach can result in inconsistencies. If they prove severe enough to the users, even inconsistencies can be ironed out.

One can imagine an extension of a system like VNET to an entire society in which individuals or groups of users could propose software facilities in return for using the software or hardware of others. Royalty credits could be paid on the basis of use, and people could use the network itself to solicit requirements from other subgroups, conduct experiments, ask for advice, and advertise potential services.

Organizational, sociological, and managerial issues become intertwined with issues of system design.

We are not recommending any particular system structure here, but merely trying to point out that as networks become more powerful, the organizational, sociological, and managerial issues become more deeply intertwined with issues of system design. The more general point to be made once again is that systems must be designed with the knowledge that they will evolve. One happy trend in this direction is the emphasis in teaching programmers to put a top priority on writing structured, readable code, rather than minimizing storage and execution speed at the expense of making a house-of-cards type of system.

Areas of needed research

Architecting for ease of use could benefit from basic research in several areas. A primary difficulty in producing a friendly system is that one cannot really predict ease of use without controlled experiments involving actual use by representative end users.

There are a number of analytic approaches that are relevant to predicting whether an interface is difficult to use. One of these approaches is Halstead's software science metrics. Halstead's basic idea was to judge

the complexity of a program, based on the numbers of unique operators and operands and the total number of operators and operands. When applied to programs varying greatly in total complexity, such metrics have a good correlation with time required and errors.[17] Halstead was beginning to extend his work to predicting the ease of use of interfaces.[18,19]

A second approach with the same goals in mind is being pursued by Phyllis Reisner at IBM San Jose Research.[20] In her approach, an interface is specified in terms of a Backus-Normal Form (BNF) description of the grammar the user needs to know in order to use the interface. Further research could extend these approaches and determine their practical utility.

Many simple natural language statements form complex queries and data base operations.

Protocols of users have been analyzed at the IBM Thomas J. Watson Research Center in Yorktown Heights, New York, and a number of fundamental cognitive problems in learning to use new computer systems have been identified.[21]

In addition to this research in cognitive science, more research is needed in several areas of computer science and computational linguistics. One such area involves techniques allowing the use of natural languages for interaction with a computer.

Many seemingly simple natural language statements turn out to be rather complex in terms of the corresponding statements in a formal query language and the data base operations required.[22] Consider the example query, "Which employees do not have a car?" A system not geared to that query would first have to find *all employees* and *all cars*, match them to identify *owners of cars*, and then subtract the owners from all employees. An inexperienced user might well have difficulty in so formalizing even this simple inquiry.

It also seems clear that systems need to be adaptive in several other ways: adaptive to particular users, adaptive to the same user over time, adaptive to the changing needs of the organization, and adaptive to changes in the hardware and software functions available on a system. Yet, how do we make systems that adapt?

Some excellent research has been done, for instance, in making a checker-playing system that learned to play better checkers through analyzing its experience.[23] And work has been done on trying to emulate some of the sophisticated genetic mechanisms for adaptation known to the biologist.

Further research remains to be done on how to apply these and other ideas in an actual operating system. One could envision adaptive staging algorithms that might look at particular users and their tasks to discover whether they have patterns of data use that would justify modifying the staging algorithms, given those conditions.

It was suggested by Carroll and Thomas[24] that metaphors are very useful as design guidelines. To the extent that a system can be explained to new users in terms of things that the user is already familiar with, the system will be more readily comprehended. Subsequent research has demonstrated the strength of metaphorical knowledge. One experimental subject, for example, in learning to use a text editor was explicitly instructed that one must backspace to erase. After reading this, the subject refused to try it, saying that backspacing does not erase. The subject tried several other actions before finally resorting to the backspace. Perhaps the classic example is the ambiguity of a word processor key with a vertical arrow that was disambiguated in instructions by the words "scroll" or "window." Yet it is rare that either of these words appears on a key.

Suggestions for education

However good the tools of the developer, and however much is known about how to design an easy-to-use system, for some time into the future there will be a considerable amount of art involved in designing a user-friendly system. Furthermore, decisions affecting ease of use will be influenced by a great many different people. For this reason, it will be very important for every programmer, architect, and documentation writer to have some appreciation of human factors as a research and design discipline. Yet most computer science and programming curricula fail in several ways.

There is rarely any explicit requirement for any courses on human factors or basic psychology. It is quite possible for a person to earn a Ph.D. in computer science, anticipating a lifetime career building tools for human beings to use, by spending several years learning how computers work, and yet no time learning how human beings work. Furthermore, the examples used in computer science courses generally do not make an explicit point about the importance of human factors, nor do they even implicitly demonstrate good human factors.

To the extent that artistic, musical, and literary people are computer-literate, the medium itself will reflect the wide range of human experience.

We suggest that textbooks and courses on system architecture specifically ask the student to consider the users and their tasks before designing the system. We also suggest that these textbooks point out the importance of human factors and illustrate good human factors in the examples.

The other side of the coin is that the texture of the computer communication tools of the future will depend upon the people who design and use them. To the extent that artistic, musical, and literary people are computer-literate and make use of this new medium, the medium itself will reflect the wide range of human experience. To the extent that the educational system isolates such people from computers, so will the medium lack those qualities. To the extent that everyone is capable of participating in communicating via computer, we will have the possibility of a more democratic society.

Several of our universities (e.g., Carnegie-Mellon University, Massachusetts Institute of Technology, and Brown University) are now engaged in rather wide-scale experiments with networks. This is being encouraged by industry and government. In fact, the universities are probably ahead of industry and business in peer open-ended human networking. This gives them an obligation to structure their activity and research so that they make a real scholarly contribution to this issue in the broadest context.

Cited references

1. B. Schneiderman, *Software Psychology*, Winthrop Publishers, Inc., Cambridge, MA (1980).
2. *Conference Proceedings*, Human Factors in Computer Systems, Gaithersburg, MD, March 1982, Institute for Computer Science and Technology, National Bureau of Standards, Washington, DC (1982).
3. S. K. Card, T. P. Moran, and A. Newell, *The Psychology of Human-Computer Interaction*, Lawrence Erlbaum Associates, Inc., Hillsdale, NJ (1983).
4. J. S. Brown and R. B. Burton, "Diagnostic models for procedural bugs in basic mathematical skills," *Cognitive Science* **2**, 155–192 (1978).
5. J. C. Thomas, "The computer as an active communication medium," *Proceedings of the Conference*, Annual Meeting of the Association for Computational Linguistics, Philadelphia (June 1980).
6. A. Chapanis, "Prelude to 2001: explorations in human communication," *American Psychologist* **26**, 949–961 (1971).
7. L. M. Branscomb, "Information: the ultimate frontier," *Science* **203**, 143–147 (1980).
8. S. R. Hiltz and M. Turoff, *The Network Nation: Human Communication Via Computer*, Addison-Wesley Publishing Co., Reading, MA (1978).
9. L. M. Branscomb, "Bringing computing to people: the broadening challenge," *Computer* **5**, No. 7, 68–75 (1982).
10. J. T. Richards and S. J. Boies, "The IBM audio distribution system," *Proceedings*, IEEE MIDCON Conference, Chicago, IL, November 10–12, 1981, IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854 (1981).
11. J. D. Gould and S. J. Boies, "Speech filing—An office system for principals," *IBM Systems Journal* **23**, No. 1, 65–81 (1984).
12. J. M. Carroll and C. Carrithers, "Training wheels on a user interface," to be published in *ACM Communications* (1984).
13. G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "Statistical semantics: analysis of the potential performance of keyword information systems," *Human Factors and Computer Systems*, J. C. Thomas and M. Schneider, Editors, Ablex Publishing Corporation, Norwood, NJ (1984).
14. G. B. Leeman, *A Formal Approach to Undo Operations in Programming Languages*, Research Report RC-10310, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1984).
15. J. M. Carroll, *Learning, Using and Designing Command Paradigms*, Research Report RC-8141, IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598 (1980).
16. J. C. Jones, *Design Methods*, John Wiley & Sons, Inc., London (1970).
17. J. M. Halstead, *Elements of Software Science*, Elsevier Science Publishing Company, Inc., New York (1977).
18. J. M. Halstead, personal communication (1977).
19. J. C. Thomas, "Psychological issues in the design of database query languages," *Designing for Human-Computer Communication*, M. S. Sime, Editor, Academic Press, Inc., London (1983).
20. P. Reisner, "Formal grammar and human factors design of an interactive graphics system," *IEEE Transactions on Software Engineering* **SE-7**, No. 2, 229–240 (1981).

21. R. L. Mack, C. H. Lewis, and J. M. Carroll, "Learning to use word processors: problems and prospects," to be published in *ACM Transactions on Office Information Systems.*
22. H. Lehmann, "Interpretation of natural language in an information system," *IBM Journal of Research and Development* **22**, No. 5, 560–572 (1978).
23. A. L. Samuel, "Some studies in machine learning using the game of checkers. II—Recent progress," *IBM Journal of Research and Development* **11**, No. 6, 601–617 (1967).
24. J. M. Carroll and J. C. Thomas, "Metaphor and the cognitive representation of computing systems," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-12**, No. 2, 107–116 (1982).

Reprint Order No. G321-5220.

Lewis M. Branscomb *IBM Corporate Headquarters, Armonk, New York 10504.* Dr. Branscomb is vice president and chief scientist of IBM and a member of the Corporate Management Board. A research physicist, he was appointed by President Carter to the National Science Board in 1979 and has been its chairman since 1980. Prior to joining IBM in 1972, Dr. Branscomb was director of the National Bureau of Standards. He joined the Bureau in 1951, served as chief of the NBS Atomic Physics Division, and was co-founder and chairman of the Joint Institute for Laboratory Astrophysics at the University of Colorado before his appointment as director of NBS in 1969. Dr. Branscomb graduated from Duke University in 1945 and earned his M.S. and Ph.D. degrees in physics at Harvard University in 1947 and 1949. A member of the National Academies of Sciences, Engineering, and Public Administration, and a former president of the American Physical Society, he has served on numerous boards and commissions concerned with science and public policy.

John C. Thomas *IBM Research Division, Thomas J. Watson Research Center, P. O. Box 218, Yorktown Heights, New York 10598.* Dr. Thomas joined IBM in 1973. He currently manages the Remote Information Access Systems group, whose goals are to produce high quality text-to-speech conversion and to study applications of speech synthesis. He spent two years on the staff of the IBM Chief Scientist, Dr. Lewis M. Branscomb. Prior to that assignment, Dr. Thomas was a Research Staff Member studying the human factors of design, Query-by-Example, and the Audio Distribution System. Before joining IBM, Dr. Thomas managed a National Institute of Mental Health grant on the psychology of aging at the Harvard Medical School. He is a licensed psychologist and has over sixty professional papers and presentations in psychology to his credit. He received his B.A. degree from Case–Western Reserve University and his Ph.D. from the University of Michigan. Dr. Thomas is a Fellow of the Institute for Rational-Emotive Therapy and an adjunct full professor at Pace University.