Human Aspects of Computing

Henry Ledgard Editor

Designing for Usability: Key Principles and What Designers Think

JOHN D. GOULD and CLAYTON LEWIS

ABSTRACT: *This article is both theoretical and empirical. Theoretically, it describes three principles of system design which we believe must be followed to produce a useful and easy to use computer system. These principles are: early and continual focus on users; empirical measurement of usage; and iterative design whereby the system (simulated, prototype, and real) is modified, tested, modified again, tested again, and the cycle is repeated again and again. This approach is contrasted to other principled design approaches, for example, get it right the first time, reliance on design guidelines. Empirically, the article presents data which show that our design principles are not always intuitive to designers; identifies the arguments which designers often offer for not using these principles—and answers them; and provides an example in which our principles have been used successfully.*

Any system designed for people to use should be easy to learn (and remember), useful, that is, contain functions people really need in their work, and be easy and pleasant to use. This article is written for people who have the responsibility and/or interest in creating computer systems (or any other systems) with these characteristics. In the first section of this article we briefly mention three principles for system design which we believe can be used to attain these goals. Our principles may seem intuitive, but system designers do not generally recommend them, as results of surveys reported in Section 2 show. The recommendations of actual designers suggest that they may sometimes think they are doing what we recommend when in fact they are not. In Section 3 we contrast some of their responses with what we have in mind to provide a fuller and clearer description of our principles. In Section 4 we consider why designers might not actually be using our design principles. In Section 5 we elaborate on the three principles, showing how they form the basis for a general methodology of design. In Section 6 we describe a successful example of using our recommended methodology in actual system design, IBM's Audio Distribution System (ADS), and the advantages that accrued as a result.

1. THE PRINCIPLES

We recommend three principles of design.

Early Focus on Users and Tasks

First, designers must understand who the users will be. This understanding is arrived at in part by directly studying their cognitive, behavioral, anthropometric, and attitudinal characteristics, and in part by studying the nature of the work expected to be accomplished.

Empirical Measurement

Second, early in the development process, intended users should actually use simulations and prototypes to carry out real work, and their performance and reactions should be observed, recorded, and analyzed.

Iterative Design

Third, when problems are found in user testing, as they will be, they must be fixed. This means design must be iterative: There must be a cycle of design, test and measure, and redesign, repeated as often as necessary.

2. WHAT SYSTEM DESIGNERS AND PROGRAMMERS ACTUALLY SAY

We began recommending these principles in the 1970's. Often the reaction is that they are obvious. Nevertheless, they are not usually employed in system design. Why? We wondered whether or not these principles were really obvious, or whether or not they just

seemed obvious once presented. To find out, during 1981–1982 we asked five groups of systems planners, designers, programmers, and developers to write down the sequence of five (or so) major steps one should go through in developing and evaluating a new computer system for end users. These people were attending a human factors talk, and did this just prior to its beginning. We suggested that they use an office system or point-of-sale terminal as an example. These 447 participants provide a particularly good test of how intuitive, obvious, regularly advocated, or regularly practiced our design principles are, for they are among the very people who design computer systems for people to use. Further, since they were attending a human factors talk, they would likely be biased to mention human factors issues. Each person's responses were graded independently by three or more judges (only one of whom was a human factors person), and disagreements were settled jointly.[1] Grading was very liberal: we gave credit for even the simplest mention relating to any one of our four principles, regardless how impoverished or incomplete the thought was.

Table I shows the key result. Most participants did not mention most of our four design principles. Twenty-six percent did not mention any of the four principles, and another 35 percent mentioned only one. Just 2 percent made any mention of all four. These percentages would have been much lower had we used a more stringent grading procedure.

As to the specific principles mentioned, 62 percent mentioned something about early focus on users; 40 percent mentioned something about empirical measurement, that is, behaviorally testing the system (or a simulation or prototype of it) on people (regardless of their characteristics); and 20 percent mentioned something about iterative design, that is, modifying the system based on these results.

The intent here is not to single out as "bad folks" all those people responsible for the creation of a system, whom we will collectively refer to as "designers." Principles of design are arguable, of course. Ours are not universal truths. Had human factors people, for example, been asked the same questions, the percents mentioning each principle might not have differed from those observed. Indeed, some other human factors people recommend design approaches that have little in common with what we recommend. This can be seen in several papers in recent conference proceedings covering the human factors of system design, for example, *The Proceedings of the Human Factors Society Meetings* [30] and *The Proceedings of CHI83 Human Factors in Computing Systems Meetings* [29].

Of course these survey results cannot be assumed to indicate what designers actually do, or would do, with real design tasks. They do show, however, that our principles are *not* obvious (at least before they are presented), consistent with the observation that they are rarely applied. Our experience is that even after hearing them, people often do not understand their full force.

TABLE I. Summary of Six Surveys in Which 447 People Attending Classes for Systems Planners, Programmers, Designers, and Developers Briefly Wrote the Key Steps One Should Go Through in Developing and Evaluating a Computer System for End Users

Percent of respondents mentioning a given number of principles:				
Number of principles	0	1	2	3
Respondents (%)	26	35	24	16

Percent of respondents mentioning each principle:		
Early focus on users	Empirical measurement	Iterative design
62	40	20

3. CONTRASTS BETWEEN WHAT WE MEAN AND WHAT WAS SAID

A closer look at the survey responses reinforces the conclusion that these "common sense" design principles are not fully understood by many designers, even when they mention them. It is our experience that people sometimes lack the ability to differentiate between what we recommend and what they do.

With respect to our survey results, there are instructive distinctions between comments which we gave credit for and what we actually recommend. In many cases, these comments may *appear* similar, but they *differ significantly* in intent, how they would be carried out, and, presumably, in their impact. Thus, at the risk of appearing overly harsh, we point out some of these distinctions to clarify what we have in mind. These distinctions are often overlooked, sometimes leading designers to believe they are following the principles that we recommend when in fact they are not.

Early Focus on Users

The design team should be user driven. We recommend *understanding* potential users, versus "identifying," "describing," "stereotyping," and "ascertaining" them, as respondents suggested. We recommend bringing the design team into *direct contact* with potential users, as opposed to hearing or reading about them through human intermediaries, or through an "examination of user profiles." We recommend interviews and discussions with potential users, and actual observations, by the design team, of users on the present version of a system. Perhaps users could try to train designers to use an existing system, and thereby designers could learn a lot about the users. (Occasionally, a proposed system will be so radical that a "present system" may not exist. We still recommend talking to the intended users, and understanding how they go about their work and what their problems are.) These interviews should be conducted *prior to system design*, instead of first designing the system and then subsequently "presenting," "reviewing," and "verifying" the design with users, or "getting users to agree" to, or to "sign off" on the design.

[1] For helping us grade these surveys, we thank Lizette Alfaro, Art Benjamin, Steve Corsaro, and Jennifer Stolarz.

As part of understanding users, this knowledge must be played against the tasks that users will be expected to perform. Other disciplines have also become aware of the absence of user involvement in design. For example, the American Association for the Advancement of Science and the National Science Foundation have established a project to address the fact that too often technologies are developed for the disabled with no input from the disabled [31].

One way to increase the saliency and importance of usability issues in designers' minds is to have a panel of expected users (e.g., secretaries) work closely with them during early formulation stages. Almost no one recommended this, not even for only brief periods of time. We call this "interactive design," and we recommend that *typical users* (e.g., bank tellers) be used, as opposed to a "group of a variety of experts" (e.g., supervisors, industrial engineers, and programmers). We recommend that these potential users become part of the design team *from the very outset* when their perspectives can have the most influence, rather than using them post hoc as part of an "analysis team (of) end user representatives." Another value of this approach, especially for the design of an in-house system, is that it allows potential users to participate in the design of a system that they will ultimately use (sometimes called "participatory design").

Some respondents recommended that potential users "review," "sign off on," or "agree" to the design before it is coded. This can be useful, but does not have the full shaping force on designers' views which an earlier association would have had. Our notion is not merely "to get users to agree" to a system design, which smacks of post hoc legalese, but to create a situation in which potential users can instill their knowledge and concern into the design process from the very beginning.

Being concerned about the "human factors of noise and light levels and safety" is important, but designers must go beyond this, understanding cognitive and emotional characteristics of users as they relate to a proposed system.

Often designers build upon previous releases (e.g., of computer systems, washing machines, cars) or add a part to an existing system. Thus, there should be little difficulty in identifying users and talking with them. We have been told that when one is at the very earliest stages of design in a new area, however, it may be hard to understand who the users will be or to interact with them. When this is so, it strengthens the arguments for empirical measurement and iterative design.

Empirical Measurement

Here we emphasize two factors: actual behavioral measurements of learnability and usability, and conducting these experimental and empirical studies very early in the development process. We gave credit for *any* mention of a user test—whether or not it was early or appropriately conceived, and even if it was suggested by context alone. Several participants who received credit for mentioning "test" seemed to have in mind a *system* test rather than a *user* test, for example, "test for system response, . . . , swapping time."

"Build(ing) a prototype to study *it* (emphasis ours) experimentally" (e.g., to study memory access speed, system reliability) is different from building a protytype to study how people will use and react to it and the training approaches and materials. It is not a question of "using a prototype to match against user requirements," but rather a question of finding out how easily people can learn and use that prototype. The first is an analytic question; the second is an empirical question. "Test(ing) the (completed) system—use it by ourselves" is good, but is not a substitute for testing it (and a series of previous prototypes) on the actual user audience.

"Reviewing" or "demonstrating" a prototype system for typical users and getting their reaction to it can result in misleading conclusions. What is required is a usability test, not a selling job. People who have developed a system think differently about its use [25], do not make the same mistakes, and use it differently from novices. Users should be given simple tasks to carry out, and their performance, thoughts, and attitudes should be recorded and analyzed.

Iterative Design

The person who wrote "make trial run of prototype and incorporate changes" makes no reference to behavioral evaluation and improvements. "Build prototype, code software, write documentation, and review" does not explicitly acknowledge the need to incorporate results of behavioral testing into the next version of the system. Finally, "if time permits, iterate the design . . ." is not sufficient or acceptable as a design philosophy. Even where iterative design was mentioned, many people seemed to feel that a single iteration or revision would be sufficient.

In answer to our question about the key steps in the development *process*, some people wrote *goals* for a system. Making a system "easy to use," "user friendly," "easy to operate," "simple," "responsive," and "flexible" are goals, indeed very difficult goals to reach. What is needed is a *process* to ultimately ensure meeting these goals. Almost no one mentioned establishing testable behavioral specifications (see below) early in the development process to see if, in fact, general behavioral goals are being met.

A Comment. One might think that it has been nitpicking or even unfair to draw upon distinctions between comments that the respondents wrote rather hastily and the points that we are trying to make. However, our experience is that these comments provide a representation of how designers of all kinds of systems (whether they are programmers of computer systems, planners of educational curriculum, authors of textbooks, architects of buildings, builders of cars, or lecturers) often think and how they view ultimate users in relation to their work. They are consistent with what other designers of computer systems say when asked how they think about design [22]. But does knowing this give us greater ability to design better systems? We

think it does because we can describe another way to do it and ask why this other way is not followed.

4. WHY THE PRINCIPLES ARE UNDERVALUED

Why do these principles seem obvious once you hear them, but do not seem to be recommended or followed in practice? The survey responses indicate that these principles are not regularly suggested and that they are not really obvious. Our experience is that they are seldom applied.

In this section we try to answer this question by identifying five categories of reasons. First, the principles may not be worth following. Second, there is confusion with similar but critically different ideas. Third, the value of interaction with users is misestimated. Fourth, competing approaches make more sense. Fifth, the principles are impractical. We see weaknesses in these reasons or objections, and we suggest ways of addressing them.

Not Worth Following

As we said earlier, principles of design are arguable, including these, and a variety of other design approaches have been recommended. Some designers, no doubt, understand our recommendations but question their value. Such objections will be resolved one way or the other as the recommendations are more fully tested in practice.

Confusion with Similar but Critically Different Ideas

It is our experience that designers often have difficulty differentiating between what we recommend and similar but critically different ideas. The survey results are consistent with this experience. Sometimes designers believe they are following what we recommend when in fact they are not. Sometimes designers confuse the intention to carry out user testing with the testing itself.

We hope these problems will resolve themselves over time. If designers have more interaction with users, and if they carry out more empirical evaluations of their work, we expect the value of these approaches, and their relationship to other methods, to become clearer.

The Value of Interaction with Users is Misestimated

User Diversity Is Underestimated. Because most designers have only limited contact with users (and this is often centered on topics of the designers own expertise and not that of the users), they simply do not realize how widely users differ, and, especially, how different many users are from most designers. If dashing off a few lines of code is trivial for a designer, then that designer is not likely to imagine that this can be extremely difficult for someone else. When users do have trouble, designers are sometimes tempted to think they are "stupid." It is difficult to give fair weight to the years of training and experience that underlie one's own ability. But more important, it is almost impossible to think about whether or not someone else will have trouble if you never encounter any yourself. In observing complete novices learning to use text editors [25] or message systems [19], we have often been amazed as they encounter major problems that we did not anticipate, or when problems that seemed simple to us were impossible for them to recover from.

User Diversity Is Overestimated. Sometimes we are told that people are so different that it makes no sense to conduct tests with only a few people. One would have to test hundreds of people and then the result would be so variable as to be useless. It is true that testing only a small sample, as is often necessary for practical reasons, cannot reveal all the problems that will arise with a design. But it is much better to identify some of the problems that some users will have than not to identify any. Further, our experience is that problems are not as idiosyncractic as is sometimes thought. The same problem, even a completely unanticipated one, often crops up for user after user.

Belief That Users Do Not Know What They Need. This objection points up a genuine problem: Getting useful design information from prospective users is not just a matter of asking. Many users have never considered alternate or improved ways of performing their tasks and are unaware of the options available for a new design. Further, in trying to communicate, designers may unwittingly intimidate users, and users may unfortunately become unresponsive.

One way around this is to present new ideas in a way that makes it easy for users to relate them to their concerns. One approach, used with a text-editing system at Wang Laboratories (personal communication, 1980), is to write a user manual and get reactions to it, as the first stage in design. Another method is to construct detailed scenarios showing exactly how key tasks would be performed with the new system, as was done for IBM's ADS [19]. Another approach is to simulate the user interface of a proposed system [21, 23]. These approaches are valuable even if no user reaction is sought: It is extremely difficult for anybody even its own designers, to understand an interface proposal, without this level of description.

Putting the design in intelligible form is not the only difficulty in getting user reaction. Users may endorse a proposal uncritically, presuming that the technical "experts" know more than they do about their needs. In the course of extended give-and-take with designers, users may come to know too much: They may understand the technical issues so well that they can no longer detect the difficulties in comprehension that others users, who do not have the benefit of weeks of dialogue with the designers, will face.

The effect of these problems is that interacting with users during design cannot in itself ensure a good design. But at least some design issues will be defined and dealt with sooner and more effectively if user knowledge is brought to bear from the start.

Belief That My Job Does Not Require It or Permit It. Sometimes organizational arrangements isolate designers from contact with users, or place the responsibility

for usability entirely elsewhere, with no role for others. Designers find themselves preoccupied with meeting a schedule for their individual system ingredient. There is no time for contact with users until their work is finished—which never quite happens. A rigid development process leaves no room for new approaches.

We have been told by a designer that it is sometimes difficult to get customers to commit productive users to spend sufficient time interacting on the design of a future system. When this is the case, designers can use techniques mentioned in this article that may require less time, for example, get reactions to an early user manual, help-line service, or printed scenarios of how the user interface might work.

Competitive necessity will eventually break down these obstacles and traditions. Good user-oriented systems cannot be built from local optimization of individual system ingredients. In the meantime, other ways to do the needed work can often be found. Small-scale usability evaluations can often be carried out without requiring much additional resource. Marketing or planning people are often eager to have development people participate in customer visits where their technical skills can be very helpful.

Competing Approaches

Belief in the Power of Reason. If system design were fundamentally a rational analysis of how a task should be done, then there would be no need to involve users. Why muddy the waters by getting information about existing, and probably irrational, practices? There are two problems with rational analysis as the sole basis of design. First, it leaves things out: Rational analysis does not tell you what you have to analyze. Here is an illustration. Some designers have been puzzled that word processing systems have not driven out the typewriter. Why do many offices have a typewriter and a word processor side by side? Does a word processor not handle all "document creation"? Just thinking logically about document creation is unlikely to reveal the key facts. But a few minutes of observation of real office work shows some of the things that document creation leaves out. Filling in forms is much easier with a typewriter. For very short documents, such as buck slips or telephone messages, the overhead of a typical word processor is unacceptable. One cannot discover the existence of these critical cases by armchair reflection on office work.

A second problem with relying only on reason is that systems almost always have to interact with preexisting work methods, and mismatches can be major problems. Even if a new system is intended to entirely replace former methods, there is still the problem of relating peoples' comprehension of the new ways to their established habits of thought. The problems surrounding this process are not subject to a priori rational analysis, but must be explored empirically, that is, by having actual users try out the new system under realistic conditions. Listening to users' comments is a good way to do this.

Rational analysis is, of course, important, for without it we are unlikely to create new innovative systems. Analytic approaches should be used when they are applicable, but they cannot be seen as a substitute for empirical methods.

Belief That Design Guidelines Should Be Sufficient. There is no handbook of operating characteristics for the human mind. Guidelines for user interface design do exist (e.g., [9]), and they can be useful. Certainly, for many designers, guidelines can help get the first version of a prototype system closer to the final desired version than if they were not used. However, they provide only general constraints on design. No matter how conscientious a designer is in finding and following this distilled wisdom, the resulting design may be very good or very bad.

One limitation of guidelines is that they cannot deal with choices that are highly dependent on context, as many of the important choices in interface design are. For example, a guideline cannot recommend that special purpose keys be used instead of typed commands because the choice depends on whether or not users are touch typists, whether or not it is possible for the system to distinguish commands from other entries if they are typed, whether or not the command set is extensible, and many other aspects of the situation. Existing guidelines are often based on informed opinion rather than data or established principles. Very few design choices have been investigated in a controlled way. Research cannot solve either of these problems in the foreseeable future.

Cognitive psychologists agree that human performance adapts strongly to the details of the task environment. We do not understand this adaptation well enough to predict the effects of most design choices in any one given situation, let alone form general conclusions about them. The same ignorance argues against conducting experiments to validate existing guidelines about which there is doubt. Feasible experiments could only investigate a choice in a few contexts, probably not increasing our confidence in generalizing about it very much. Psychology is not close to being able to develop significantly improved guidelines to overcome these limitations. Human factors can provide a *process* by which usable and useful systems can be designed, but cannot provide design guidelines in enough detail to determine how a system should ultimately appear to users. We feel, at present, that guidelines should be viewed as an informal collection of suggetions, rather than as distilled science. Designers will have to make many choices on their own, and be prepared to test their work empirically.

Belief That Good Design Means Getting It Right the First Time. "Getting it right the first time" seems like a laudable goal, and is, in fact, an alternative design philosophy to our own; but experience shows it is not achievable in user interface design. Certainly careful design work pays off, and the need to iterate is not a license to be sloppy. Assuming that iteration will not be

needed, when laying out a schedule and choosing implementation methods, is asking for disaster in user interface design. Even the "zero defects" approach, developed by Crosby [6] for general quality control, advocates the need for evaluative testing and empirical measurement. It does not simply assert that one can, from the outset, create a design for zero defects.

"Getting it right the first time" plays a very different role in software design which does not involve user interfaces than it does in user interface design. This may explain, in part, the reluctance of designers to relinquish it as a fundamental aim. In the design of a compiler module, for example, the exact behavior of the code is or should be open to rational analysis. Even those factors which cannot be predicted exactly, such as frequencies of data with particular characteristics, may be amenable to statistical treatment. The choice of algorithms can and should be anchored securely in a reliable analysis of the data, the transformations to be carried out, and the performance characteristics of the system. Good design in this context is highly analytic, and emphasizes careful planning. Designers know this.

Adding a human interface to the system disrupts this picture fundamentally. A coprocessor of largely unpredictable behavior (i.e., a human user) has been added, and the system's algorithms have to mesh with it. There is no data sheet on this coprocessor, so one is forced to abandon the idea that one can design one's algorithms from first principles. An empirical approach is essential. The involvement of human users escalates the need for an empirical approach well above the usual requirements for testing to make sure a system works.

When large diverse groups are involved in developing a system, we have observed a practice of "freezing the user interface" early in the development process (even prior to coding it). Apparently this reflects the need to have some aspect of the system fixed or stable as the various groups then proceed somewhat independently with their own work. But the user interface is exactly that part of the system which should be open to change. The best this approach can achieve is that a system can get *programmed* in an error-free manner, not that the resulting interface will be of high quality. It is impossible to design the system right the first time because this is based on the assumption of a perfect *forecast* of the best user interface—something which can only be determined empirically. Further, fixing the user interface early assumes nothing will be learned over the next two years, or so, of development.

When one is an outside contractor (rather than in an internal system development organization), it is often difficult to get a customer to sign a contract that includes the flexibility required in iterative design. There is, typically, insistence, we are told, on a two-stage, "preliminary design" and "final design" hierarchy, with schedule rigidity that often precludes proper accommodation of usability tests results. Ignoring the need for iterative design is perhaps even more disastrous here since geographic remoteness may further reduce required communication and observations needed to attain good usability.

Our system design philosophy is neutral vis-a-vis some other well-known strategies for program design, for example, top-down design [7], top-down testing, or structured programming [8]. Yourdon and Constantine [35] have reviewed these and other programming design strategies. The small group aspect of chief programmer teams [1] is important, we believe, in providing consistency and simplicity in overall system usage.

Impractical

Belief That the Development Process Will Be Lengthened. In a competitive world, products are always developed under time pressure. Schedules are critical. Designers sometimes fear that their schedules will not be met if behavioral testing is done. Will the development process not be lengthened by creating a prototype? Will the development process not be lengthened further by doing user tests with it? Will the development process not be lengthened even further by redesigning on the basis of the user results? We feel that these questions reflect two underlying assumptions. The first is that usability work must be added to the end of the development cycle, as opposed to overlapped with it. The second is that responding to tests must be time consuming.

With respect to this first assumption, one can do user testing before a system is built, and continue this work throughout the development process. One can create paper and pencil tasks that test critical features of the interface such as the syntax of commands. IBM's "Query-by-Example" [36] was evaluated by asking users to write down the queries they would construct to answer questions given to them in English [33]. This was done before a single line of code was written. It was therefore done without the benefit of system feedback [33] which was studied in later experiments [3, 5]. More comprehensive testing can be done by constructing a simulated system. For example Gould, Conti, and Hovanyecz [21] did extensive testing of a "listening typewriter," a device that would use speech recognition to give real-time visual feedback during dictation, by using a human typist in the computer feedback loop. Kelley [23] used a computerized simulation of a calendaring system in which the experimenter could invisibly enter the user-system dialogue whenever the computerized system would not give an appropriate response. Here, again, both of these simulations were done before a line of code was written for the real systems. Once a running prototype exists, experimental tests with real users can be conducted, followed by empirical (field) studies of training, user interface, and reading materials used together.

It is our personal experience and observation that building simulated or informal prototypes, rather than delaying or lengthening system development, actually helps get a new project off the ground, gives it something tangible for others to see, and stimulates thought and progress.

We have been told that with some new systems the main issue is sometimes one of technical feasibility or capability of the technology to perform in a certain way. "How can this be explored without building a box?" we have been asked. The answer is that is exactly what was done in the Thomas and Gould [33], the Gould, Conti, and Hovanyecz [21], and the Kelley [23] simulation studies. While some aspects of new technology may be difficult to simulate we have never encountered a design problem in which at least some important aspects could not be usefully simulated.

With respect to the second assumption, that responding to the user test results must be time consuming and expensive, it is possible to build a system so that one can do this quickly and easily. The implementation is structured so that the user interface can be changed without changing the implementation of underlying services. In a sense, the system becomes its own prototype, in that it is easy to construct and evaluate alternative designs. IBM's ADS, discussed in more detail below, has this structure.

Even when these approaches are taken, there is no denying that user testing still has a price. It is nowhere near as high as is commonly supposed, however, and it is a mistake to imagine that one can save by not paying this price. User testing will happen anyway: If it is not done in the developer's lab, it will be done in the customer's office. Brooks [4] has pointed out that everyone builds a prototype. The only question is whether or not, in the case of vendors, they also market it as a product, or in the case of in-house development, they give it to their users. The price is poor quality, extra (unanticipated) customer expense, and extra (and unanticipated) vendor costs. Changes that must be made after the product is delivered are, of course, much more expensive than those made even late in development. They must be done piecemeal, and under more constraints of compatibility, in that changes have to be minimized to avoid disrupting users. Fixes are likely to be superficial, and quality will continue to suffer. Heavy reliance on initial customer feedback, rather than early empirical simulations, prevents innovation because too many constraints then exist, making fresh substantially different approaches impossible.

Belief That Iteration Is Just Expensive Fine-Tuning. Our philosophy is not just a trivial expensive matter of "fine-tuning," but a basic design philosophy to be contrasted with other principled design philosophies. An iterative design philosophy may seem expensive, but with the present state of understanding about user interface design, it is the only way to ensure excellent systems. The three principles we outlined can be extended and coordinated to form an overall approach to user interface development, as is partially done in the next section.

Belief in the Power of Technology to Succeed. We have been told that technical people have a lot of faith in the "power of technology" to succeed. People will buy it in spite of the user interface. This has been true at the high end of computer systems, and was true in the case of the automobile industry. But as the American automobile industry found out, other manufacturers will make the necessary accommodations to users. We belive the same thing will happen in the computer industry. Just because there is a speech recognition system, a touch screen, a wireless terminal, or a picture phone is no longer a guarantee that these will succeed. Increasingly, with computer systems the product *is* the user interface. This reinforces the points we are trying to make. More and better students are becoming involved with the human factors of computer systems, and they will be developing new methodologies and providing a stream of findings on usability, which may very well exert powerful effects in the marketplace.

5. AN ELABORATION OF THE PRINCIPLES

To carry out our suggestions, we roughly divide the activities required in explaining our recommended principles into an initial design phase and an iterative development phase, although there is no sharp dividing line separating them.

Initial Design Phase

Preliminary Specification of the User Interface. This is only one of several activities that need to be attacked early. Here are others.

Collect Critical Information About Users. Some of what is needed, such as literacy level or how long users stay at one job (both of which affect training requirements), can sometimes be gathered second-hand, from surveys or consultants. But direct contact with potential users is essential to flesh out the basics. Reluctance or willingness on the part of the users to read manuals, tolerance for delay or effort, and expectations about what a new system should provide are examples of factors that are unlikely to come through in second-hand descriptions of users but which designers need a feel for. Perhaps more important, one does not know what one needs to know about a user until one sees the user in person. These contacts are almost always full of surprises.

Sometimes there is a (understandable) tendency for designers to want to look up in a book what the characteristics of a class of users (e.g., bank tellers) are (an extension of the guidelines approach), and build a system from these. We have tried to find an example of a system whose user set is so narrow and so well specified that general user characteristics, such as reading level, age, and so forth, would be an adequate basis for design. We have not found any. To the extent that the scope of users and tasks becomes broader, understanding the user becomes all of psychology (cognitive, behavioral, anthropometric, attitudinal, etc. characteristics), and general descriptive data will be of even less value.

As noted earlier, one of the surprises may be how difficult seemingly easy operations may really be for users. Direct contact with users, both in this phase and

in later behavioral testing, can make designers aware of just where these difficulties are cropping up.

There is an analogy between the sort of insight into users and their needs that a designer must have and the sort of insight into the needs of a particular industry that a software developer must have. Just as "insider" knowledge is essential to develop really useful software for banking, say, or insurance applications, so an "inside" view of user requirements is essential to create a superior user interface. For most designers the only way to get this inside view is by close consultation with users.

Such consultation is greatly facilitated if the users can see and react to a real "users'-eye-view" of the proposed system. This can be done by preparing a users manual for review, as has been done at Wang for a word processor (personal communication, 1980), by presenting detailed usage scenarios, as was done for ADS [19], or possibly by presenting a description of how a user would interact with the system, as was done at Apple for the Lisa computer system [34]. Even if it is not used in user consultations, preparing such a users view can be helpful in focusing design energy on interface issues. It can also form the basis for behavioral specifications and tests.

Develop Behavioral Goals. The plan for a new system always includes performance and capacity targets for the hardware and software, for example, memory size and access rates, and calculation times. These need to be supplemented by targets which specify how well the *user* must be able to perform using the system. For example, one might specify that 80 percent of a sample of representative users must be able to master specified basic operations on a word processor in half an hour. With such goals in place it is possible to consider whether or not proposed design features or design changes will contribute to the goals. Without such goals, it is easy for such issues as implementation convenience or memory requirements to dominate the design to the detriment of usability. Thus, when viewed properly, a major reason for behavioral targets is that they provide a management tool to assure that system development proceeds properly.

Behavioral goals should be testable, that is, there should be a clear procedure for determining whether or not a design meets the goals. This will mean that the statement of the goals must cover at least the following points.

1. A description of the intended users must be given, and the experimental participants to be used to represent these users in tests should be agreed upon: for example, typists supplied by temporary employment agencies in Los Angeles (30 percent of whom have English as a second language).

2. The tasks to be performed, and the circumstances in which they should be performed, must be given. For example, a test scenario might specify that the participant will be given a manuscript and asked to use a prototype copier to make five copies on legal size paper (not presently in the copier), collated and stapled. No assistance would be available except a telephone "hot line."

3. The measurements of interest, such as learning time, errors, number of requests for help, or attitude, and the criterion values to be achieved for each, must be given. Most systems are improvements on older ones, and in these cases it is relatively easy to specify the behavioral criteria, for example, learning time. But it is harder to establish the appropriate values these criteria must take on, and this may have to be done iteratively. In the case of an altogether new system, where the functions have not previously been implemented, specifying the criteria correctly the first time may also be hard, and iteration will be required.

Any specifications, including behavioral goals, influence the design process in complicated ways. Rigid enforcement of specifications is often impossible, but even when they are violated, specifications help to focus design attention and effort in the right places. The process of creating and agreeing on a good set of specifications can be valuable in itself. This process can help clarify the validity of various measures of usability.

Organize the Work. The user interface of a system is a complex entity with diverse parts. The software, the workstation from which the software is operated, the training procedure (if any) in which users participate, the reference manuals or materials, all work or fail to work together to create the conception with which the user ultimately deals. Unfortunately these interacting pieces are usually designed separately. Definers, designers, implementers, application writers, and manual writers constitute large groups in themselves, and are often separated by geography or organization. They often become part of the development process at different times, and thus must accept what earlier participants have already solidified. The picture can be even worse when significant work, such as writing user manuals, is vended out to third parties. It appears that superior quality can be attained only when the entire user interface, including software, manuals, etc., can be designed by a single group, in a way that reflects users' needs, and then evaluated and tuned as an integrated whole. This approach was followed with ADS, as discussed below.

Iterative Development Phase

With testable behavioral goals, and ready access to user feedback, continuous evaluation and modification of the interface can be undertaken. But it will only be feasible if an implementation strategy that permits early testing of design features and cheap modification of the evolving implementation has been planned. Such a strategy has to include fast flexible prototyping and highly modular implementation. These virtues can be combined: The ADS system was essentially self-prototyping, in that the final implementation, in fact, has the structure of a prototyping tool, with table-driven interface specification. This approach solved two problems often associated with prototyping. First, little work was

invested in a separate prototyping system that was then discarded. Second, once design features were prototyped there was no further work needed to incorporate them in the final implementation since the prototype and final implementation were the same.

Experience shows that iterative design should not be thought of as a luxury tuning method that puts finishing touches on a design (at great expense). Rather, it is a way of confronting the reality of unpredictable user needs and behaviors that can lead to sweeping and fundamental changes in a design. User testing will often show that even carefully thought out design ideas are simply inadequate. This means that the flexibility of the implementation approach has to extend as far into the system as possible. It also means that designers have to be prepared for evaluation results that dictate radical change, and must have the commitment to abandon old ideas and pursue new ones. Prototype testing can identify system problems with reliability and responsiveness. These two factors are absolutely necessary for a good user interface and interact with other usability factors.

We have already mentioned methods to determine whether or not behavioral targets are being met. When behavioral targets are not being met, how does one find a remedy? This is usually a very tough problem. Often user comments are the best source of ideas since they may reveal why particular errors are occurring. For example, user comments can quickly show that particular wording on a screen or in a manual is unfamiliar and is being misinterpreted. It may be desirable to collect comments while the user is working with the system since impressions given after a task is complete are often sketchy and may gloss over difficulties that were eventually overcome. The "thinking-aloud" technique, borrowed from cognitive psychology [10, 24, 27] can be useful in such cases. Of course such methods may not be appropriate in assessing whether or not a behavioral goal is being met since the process of collecting comments may interfere with or artificially improve users' performance with the system. But the problem of determining *whether or not* behavioral goals are being met is different from deciding *why* they are not being met, and what to do about it. Different methods are needed for these two aspects of the evaluation process.

A Comment. Some readers may feel that our recommendations are "not science." They may be disappointed that we do not, for example, enthusiastically describe recent developments in cognitive psychology as being able to predict design details for a new user interface or for user reading material. However, design by its very nature is not just science, but also involves engineering, history and custom, art, and invention. Our recommended approach is the best way to develop the scientific aspects of the human factors of system design. This is so for two reasons. First, the methodologies available are sufficiently rigorous and conform to the traditional scientific approach. Within the framework we outline, the methodologies range from the pure observation, analysis, and hypothesis testing of ethologists to psychophysics so precise that no man-made system can measure as accurately. Second, the approach we recommend ensures that real situations and problems will be studied, in their full complexity. This enables talented designers, human factors people, and management to identify and concentrate on the critical problems that must be solved to produce superior usability.

6. A CASE STUDY—IBM'S AUDIO DISTRIBUTION SYSTEM

As compared to the methods of science, much less is known and written about the processes of technology development [11]. Generally, the development process for most systems is (understandably) kept confidential, or at least not often written about. The exceptions, such as the interviews with designers of Lisa [26] can be very instructive. We offer here a short summary of the development of the IBM Audio Distribution System, called ADS, emphasizing the action of the design principles we have presented. In practice, actual development of a system follows any design philosophy only approximately, regardless of how formal or precisely mandated it is. Goals evolve as new ways of doing things are figured out and new useful functions are identified. ADS was no exception.

ADS is a computer-based message system that allows users to send and receive messages using a touch-tone phone as a terminal [19, 20]. Such functions as reviewing previously received messages, creating and using distribution lists, inquiring as to whether or not a message has been heard by the recipient, and changing passwords are all performed by choices and commands entered on the pushbutton telephone. ADS was intended to be used by people with no other exposure to computers, with minimal training. Ease of learning and use were paramount among design goals. Evidence to date indicates that it is very easy to learn. Customers report new users are often able to learn ADS with no training. The principles presented in this article partially evolved from the experience gained in meeting these goals.

Early Focus on Users

The target population was identified very early: managers and professional people. It was known that these people typically do not use computers themselves and do not have computer terminals. They travel frequently, and work in many different places, so access to the system away from the office is important. These considerations led to an emphasis on the use of an ordinary pushbutton telephone as a terminal, even though it was clear that restricted keypad and lack of visual output would be tough constraints.

It was also recognized that these people would be discretionary users, in that they would not be required to use ADS, but would only use it if it seemed sufficiently easy and useful to do so. They indicated that they would spend little time or effort learning a system. This led to very great effort directed toward making the

user interface as self-explanatory as possible, and matching the functions as closely as possible to user needs.

The initial set of functions designed into ADS were quite different from those which eventually emerged [19, 20]. Initially the system was thought of mainly as an enhanced dictation system, in which dictated memos could be filed and retrieved, and routed to a transcription center. Secondarily, ADS was initially thought of as an "electronic mail" communication system for relatively brief spoken messages. Laboratory experiments began to indicate that dictating was not as efficient a composition system as originally thought, and that speaking was a potentially superior composition method [12, 13, 15, 16, 19, 20]. Only after a prototype was in use was it determined that the spoken message communication features of the system were the really useful ones, however. The dictation transcription feature was then deemphasized.

This example illustrates several points we have tried to make. First, initial interaction with users did not start as early with ADS as we would now suggest it should. As a result, the first command language was cumbersome. Second, even when early interactions with users did take place, they often could not say what would be useful new functions for them. Almost no one envisioned how useful noninteractive voice communication would be. Third, giving potential users simulations and prototypes to work with enhanced the quality of feedback they gave. Empirical prototype studies identified, for example, which functions were actually used. Fourth, the architecture (or programming technology), and the designers' motivation, was flexible enough to allow iterative design.

The prototype system led to extensive interaction between users and designers. Users were free with suggestions about what they did not like (such as pushing a lot of keys to accomplish basic functions, having to remember the digits for specific commands, for example, *71 to Record, the necessity to read documentation or* spend time for training, and what they thought should be added). Having a Pending Message Box to remind the sender and recipient that an action is needed was based on a user suggestion.

Empirical Measurement

Throughout the development of the system, a great many different forms of user testing were used. Most concentrated on the ability or inability of test users to learn to perform a given set of functions with a given form of training or prompting. Some tests used simple paper-and-pencil methods, in which users had to write down the keys they would use to perform a task. Other tests involved watching users use a keypad, writing down and video-taping what they did; still others involved memorization and recall studies of various command possibilities; laboratory experiments on spoken message quality [28]; and experiments on impression formation [32]. Studies of new users almost always evaluated a combination of training, reading materials, and user interface. Versions of prototype systems in actual usage were demonstrated to visitors and at technical meetings for several years. This provided useful feedback. This work was carried out by Stephen Boies, John Conti, John Gould, Nancy Grischkowsky, Don Nix, and John Thomas. These tests led directly to many changes in the wording of messages, the organization of commands, the style of training, and other aspects of the system [19].

Later, a simple but flexible simulation tool in which a subset of keys on a computer terminal modeled the pushbutton telephone keypad was developed. Prompts and messages were automatically shown on a screen where an experimenter could read them to the test user. The action of the simulator was easily changed without programming. The experimenter could edit a set of tables that determined what would happen when a given key was pressed in a given state of the system. These tables were designed and implemented by Stephen Boies and John Richards, and an illustration is given in Table II.

Iterative Design

This simulator proved so useful that it was eventually incorporated as the actual user interface of the system itself. That is, the operation of the ADS system now marketed by IBM is controlled by tables identical to those used in "programming" the simulator. This means that sweeping changes to the user interface of the actual system can be made with no reprogramming whatsoever, simply by editing the control tables.

Once in place, this feature of the system was exploited to the full. When the system was prepared for release as an IBM product, user testing was continued until very late in the development cycle since changes were so easy to incorporate. It proved possible to include three separate user interfaces, designed for different user classes, in the product, since specifying an interface was so well isolated from the rest of the product.

What were some of the changes that all this flexibility made possible? One is a good example of the small but critical corrections that are so hard to spot in advance of testing. In one well-tested version, *R* (the 7-key) was used for RECORD and *T* (the 8-key) was used for TRANSMITting a message. This was satisfactory, and was in general use for over a year. As part of a major redesign to add new functions, it was felt that *S* (the 7-key) for SEND and *T* (the 8-key) for TALK provided a more natural-sounding command set. What could be more natural? Several months of informal user testing revealed a problem: When using this new command set users tried to SEND a message before TALKing it. (In the other case, users almost never tried to TRANSMIT a message before RECORDing it.) "I want to SEND a message to Smith," a user would reason. It was not clear that they had to TALK a message before SENDing it because SEND seemed to mean the whole action of composing and transmitting a message, at least for many novice users. Changing *S* for SEND to *T* for

TABLE II. An Example of a "Standard Table"

TNEUT	HEAD	LVL1 + LVL0, TNEUT, 0, 0, 3000	NEUTRAL MODE
	LINE	0, 0, NONE, EMPTY, 0, 0	NOT USED IN THIS TABLE
	LINE	1, 0, NONE, EMPTY, 0, 6	NOT USED IN THIS TABLE
	LINE	2, 0, COSLINE, TCUST, 0, 12	CUSTOMIZE MODE
	LINE	3, 0, NONE, TDISC, 0, 0	FAST DISCONNECT
	LINE	4, 0, COSLINE, XGET, 0, 0	GET MODE
	LINE	5, 0, COSLINE, XLIST, 0, 0	LISTEN AND EDIT
	LINE	6, 0, NONE, EMPTY, 0, 6	UNDEFINED KEY
	LINE	7, 0, COSLINE, XRECD, 0, 0	RECORD MODE
	LINE	8, 0, COSLINE, XXMIT, 0, 0	TRANSMIT MODE
	LINE	9, 0, NONE, EMPTY, 0, 0	NOT USED IN THIS TABLE
	LINE	*, 0, NONE, EMPTY, 0, 5	STAYS IN NEUTRAL: OK
	LINE	OPER, 0, NONE, EMPTY, 0, 6	NOT USED IN THIS TABLE
	LINE	#, 0, NONE, QNEUT, 0, 0	TELL USER WHAT TO DO
	LINE	DELAY, 0, NONE, QNEUT, 0, 0	TELL USER WHAT TO DO
	LINE	EOM, 0, NONE, EMPTY, 0, 0	NOT USED IN THIS TABLE

Note: Lines beginning with LINE 1 through LINE # correspond to the keys on a pushbutton telephone. If a user presses one of these keys, the corresponding LINE is executed in the table. For example, if a user presses 2 (i.e., the C-key) to customize his or her profile, LINE 2 is executed. That is, system message 12 is played out ("Customize"), a routine called COSLINE is called to initialize some variables, and control is transferred to a table called TCUST. If the user fails to push any key within 30 seconds (i.e., the 3000 centiseconds specified in HEAD) after arriving in this table, then LINE DELAY is executed. This will transfer control to a table called QNEUT which in turn will select a helpful prompt for the user, based on what is appropriate for the user to do.

TRANSMIT fixed the problem. Note that TRANSMIT is a less common, more technical term: Guidelines would probably rule against it (although some recent evidence is consistent with it; [2]). But the empirical method made the right call over our rational analysis.

Another example had to do with users making modifications to a message they were listening to. ADS asked users whether or not they wanted to add a comment at the beginning of the message, add a comment where they had stopped listening, or erase the message and start over. Some new users had trouble with this concept. For example, the wording "add a comment . . ." made sense if they were listening to a message from someone else, but not if they were listening to a message they were composing themselves. On the other hand, ". . . start over" made sense for messages they were composing themselves. Yet all three alternatives were important for both cases, for example, users needed to "insert" in their own messages (rather than "annotate" or "comment"). After testing many alternative wordings on many first-time users (which gave insight into the problem), ADS tables were "reprogrammed" to play out a slightly different prompt depending on whether users were listening to a message from someone else or one that they had composed themselves. This was easy to do at the level of the tables but required a fundamental algorithm modification so that ADS would distinguish between these two types of messages.

In the earliest stages of ADS, there were no specific behavioral goals. It was intended that the system be "easy to use," "useful," etc. We had not yet developed the principled type of thinking outlined in this article. With time, however, one behavioral goal was to create a system which required no user training. For several years, informal tests on possible user interface changes were motivated by this goal, and each major prototype revision reflected this goal. The command language was reorganized and emphasis on documentation was modified greatly. Informal feedback from customers and users indicates that a majority of new users learn ADS with no training, which is radically different from what was found for the earliest ADS prototype and for new users of most computer systems today.

Beyond ADS

It may seem that ADS is an unfair example of the application of our design ideas. The very simple terminal, with limited input and output, lent itself very well to table-driven design, with the flexibility that it provides. It was developed by a small group of people, several of whom had behavioral expertise. Could the same approaches work with a more typical system?

We believe they can. The key lesson of the ADS experience is not the implementation strategy—that is the secondary lesson. The most important lesson is the unpredictability of good design: The large number of features of the final design that were not and could not have been anticipated in the initial design. These features were only discovered and incorporated because of the focus on users and user testing in the design process.

The implementation strategy played a supporting role: It made it possible to respond to user feedback quickly and cheaply. Further, it gave *real control* of the user interface to the people who had responsibility for usability. No longer did they have to get systems experts to devote extensive time to making simple

changes in the user interface. While table-driven implementation may not be possible in some cases, the underlying idea can still be used. One approach is to identify the system functions and a set of high-level interface services that control such things as the positioning of information on a screen, collecting user responses, and the like. All these are embodied in a set of routines or macros. The interface designer can now program the interface at a high level and make changes freely without reprogramming any of the underlying services.

7. CONCLUSIONS

Computer systems are hard for most people to learn and use today. We believe that if systems were designed using the three principles we have mentioned, they would receive much higher usability marks. Survey data show that these principles (early focus on users, empirical measurement, and iterative design) are not intuitive. There is one case history, and parts of others, which indicate that the principles lead to usable systems.

Acknowledgments. For their comments on an earlier version of this manuscript, we thank Karen Assunto, Dick Berry, Stephen Boies, Jack Carroll, Robin Davies, John Hughes, John Karat, Jeff Kelley, Emmett McTeague, John Richards, Bob Taylor, and John Thomas. Some of our ideas were developed as a result of working on the ADS project and were influenced by Stephen Boies. Others were developed or sharpened while conducting a study group on human factors at IBM in 1979. Other members of that group were Jack Carroll, Web Howard, John Morrisey, and Phylis Reisner.

REFERENCES

NOTE: References 14, 17, and 18 are unreferenced in the text.

1. Baker, F.T., and Mills, H.D. Chief programmer teams. *Datamation*, (Dec. 1973), 58–61.
2. Black, J., and Moran, T. Learning and remembering command names. In *Proceedings of the Human Factors in Computer Systems Meetings*. (Gaithersburg, Md.), ACM, Washington, D.C., 1982, 8–11.
3. Boyle, J.M., Bury, K.F., and Evey, R.J. Two studies evaluating learning and use of QBE and SQL. Tech. Rep. HFC-39, IBM GPD Human Factors Center, San Jose, Calif., 1981.
4. Brooks, F.P. *The Mythical Man-Month: Essays on Software Engineering*. Addison-Wesley, Reading, Mass., 1975.
5. Bury, K.F., and Boyle, J.M. An on-line experimental comparison of two simulated record selection languages. In *Proceedings of the Human Factors Society Annual Meeting*, (Seattle, Wash.), R.E. Edwards, (Ed.), 74–78, 1982. (Available from the Human Factors Society, Box 1369, Santa Monica, Calif. 90406).
6. Crosby, P.B. *Quality is Free*. New American Library, New York, 1979.
7. Dijkstra, E.W. *Structured Programming: Software Engineering Techniques*, NATO Scientific Affairs Division, Brussels 39, Belgium, Apr. 1970, 84–88.
8. Dijkstra, E.W., and Hoare, D. *Structured Programming*. Academic Press, N.Y., 1973.
9. Engel, S., and Granda, R. Guidelines for man/display interfaces. Tech. Rep. TR00.2720, IBM, Poughkeepsie Lab., N.Y., 1975.
10. Ericsson, K.A., and Simon, H.A. Verbal reports as data. *Psychol. Rev. 87*, (1980), 215–251.
11. Gomory, R.E. Technology development. *Science 220*, (1983), 576–580.
12. Gould, J.D. An experimental study of writing, dictating, and speaking. In *Attention and Performance VII*, J. Requin, (Ed.), Erlbaum, Hillsdale, N.J., 1978, 299–319.
13. Gould, J.D. How experts dictate. *J. Exp. Psychol.: Hum. Percept. Perform. 4*, 4 (1978), 648–661.
14. Gould, J. D. Experiments on composing letters: Some facts, some myths, and some observations. In *Cognitive Processes in Writing*. L. Gregg, and I. Steinberg, (Eds.) Erlbaum, Hillsdale, N.J., 1980, pp. 98–127.
15. Gould, J.D. Composing letters with computer-based text editors. *Hum. Fact. 23*, (1981), 593–606.
16. Gould, J.D. Writing and speaking letters and messages. *Int. J. Man Mach. Stud. 16*, (1982), 147–171.
17. Gould, J.D., and Boies, S.J. How authors think about their writing, dictating, and speaking. *Hum. Fact. 20*, (1978), 495–505.
18. Gould, J.D., and Boies, S.J. Writing, dictating, and speaking letters. *Science 201*, (1978), 1145–1147.
19. Gould, J.D., and Boies, S.J. Human factors challenges in creating a principal support office system—The speech filing system approach. *ACM Trans. Office Inform. Syst. 1*, 4 (1983), 273–298.
20. Gould, J.D., and Boies, S.J. Speech filing—An office system for principals. *IBM Syst. J. 23*, (1984), 65–81.
21. Gould, J. D., Conti, J., and Hovanyecz, T. Composing letters with a simulated listening typewriter. *Commun. ACM 26*, 4 (1983), 295–308.
22. Hammond, N., Jorgensen, A., MacLea A., Barnard, P., and Long, J. Design practice and interface usability: Evidence from interviews with designers. In *Proceedings of the CHI83 Human Factors in Computing Systems* (Boston, Mass., Dec. 1983), ACM, N.Y., 40–44.
23. Kelley, J.F. Natural language and computers: Six empirical steps for writing an easy-to-use computer application. Ph.D. dissertation, Johns Hopkins University, 1983. (Available from University Microfilm International; 300 North Zeeb Rd. Ann Arbor, Mich. 48106).
24. Lewis, C.H. *Using the "thinking aloud" method in cognitive interface design*. IBM Res. Rep. RC-9265, Yorktown Heights, N.Y., 1982.
25. Mack, R., Lewis, C.H., and Carroll, J. Learning to use word processors: Problems and prospects. *ACM Trans. Office Inform. Syst. 1*, 3 (1983), 254–271.
26. Morgan, C., Williams, G., and Lemmons, P. An interview with Wayne Rosing, Bruce Daniels, and Larry Tesler. *Byte*, 1983, 90–113.
27. Newell, A., and Simon, H.A. *Human Problem Solving*. Prentice-Hall, Englewood Cliffs, N.J., 1972.
28. Nix, D. Two experiments on the comprehensibility of pause-depleted speech. IBM Res. Rep. RC-6305, Yorktown Heights, N.Y., 1976.
29. *Proceedings of the Human Factors in Computing Systems Meetings*. (Washington, Mar. 1981; Boston, Dec., 1983) (Available from ACM, Box 64145, Baltimore, Md. 21264).
30. *Proceedings of the Human Factors Society Meeting*, (Seattle, Wash, Oct. 1982; Norfolk, Va., Oct., 1983) (Available from the Human Factors Society, Box 1369, Santa Monica, Calif. 90406).
31. *Science*. New Project Explores Disability Research, *233*, (1984), 157.
32. Thomas, J.C. Office communications studies: I. Effects of communication behavior on the perception of described persons. IBM Res. Rep. RC-7572, Yorktown Heights, N.Y., 1979.
33. Thomas, J.C., and Gould, J.D. A psychological study of query-by-example. In *Proceedings of 1975 National Computer Conference*. (1975), 439–445.
34. Williams, G. The Lisa computer system. *Byte* (1983), 33–50.
35. Yourdon, E., and Constantine, L.L. *Structured Design*. Yourdon, New York, 1976.
36. Zloof, M.M. Query by example—A data base language. *IBM Syst. J. 4*, (1977), 324–343.

CR Categories and Subject Descriptors: H.1.2 [**Models and Principles**]: Users/Slash Machine Systems—*human factors*; D.2.2 [**Software Engineering**]: Tools and Techniques—*user interfaces*; D.2.9 [**Software Engineering**]: Management—*software quality assurance (SQA)*

General Terms: Human Factors

Additional Key Words and Phrases: systems development, principles of design.

Received 3/84; revised 9/84; accepted 10/84

Authors' Present Addresses: John D. Gould, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598. Clayton Lewis, Department of Computer Science, ECOT 7-7 Engineering Center, Campus Box 430, Boulder, CO 80309.