HOW TO DESIGN USABLE SYSTEMS
(EXCERPT)

John D. Gould
P. O. Box 704
IBM Research Center -- Hawthorne
Yorktown Heights, New York 10598

ABSTRACT

This paper describes a process of system design that, if you follow it, will help you design good computer systems for people -- systems that are easy to learn, easy to use, contain the right functions, and are liked. There are four key points in this usability design process: early focus on users; empirical measurement; iterative design; and integrated design, where all aspects of usability evolve together from the start. This paper focuses upon 20-30 informal methods to carry out these four points. Many of the methods can be used without extensive training in human factors. This is important because usability is so broad and so deep that there is not enough trained human factors people to work on it. Use of these methods is contrasted with other approaches frequently discussed today, including user interface standards, handbooks and guidelines, and formal models.

INTRODUCTION

The focus of this paper is on designing computer systems. It is aimed at system designers who want to know more about how to design useful, usable, desirable computer systems -- ones that people can easily learn, that contain the functions that allow people to do the things they want to do, and that are well liked. Throughout, "good computer systems" mean ones with these characteristics.

The intended audience, or users, of this paper are expected to be (1) human factors people involved, or getting involved, in system design; (2) experimental psychologists and other discipline-oriented people who may want to learn how to do behavioral system design work; and (3) system designers who are not trained in human factors but who are concerned about usability of systems. This latter group is particularly important since it is impractical for trained human factors people to work on all aspects of usability. There are not enough of them. Usability is too broad. And, besides, as a result of reading this paper, systems people can learn how usability people ought to work.

Reprinted, with some additions, from Martin Helander (Ed.), Handbook of Human-Computer Interaction. North-Holland: Elsevier, 1988, pp 757-789.

"There is no comprehensive and generally accepted manual on how to design good human factors into computer systems," wrote Shackel (1984). While not comprehensive, we hope this paper will at least be useful. It is intended to be tutorial, to identify and explain the main things you must do to design a good system. It provides indications of where to go for more information, and what you can expect to find when you get there. It is not a critical review of the literature, nor an examination of small technical differences. It brings together material from diverse sources that previously may have existed fairly independently of each other.

Overview

In this Introduction, we first note that usability consists of many pieces. Second, we briefly mention a process for system design which addresses all these pieces. We believe that the four key points ("principles") of this process are the required cornerstones for designing good systems. Third, as a means of showing where these principles fit into system design, we divide the design process into four rough phases. Next, in the main section we describe many methods to be used in carrying out this process of designing for usability. In sections at the end, we discuss tips on how to get started designing your system and various resources available to you; and we describe in more detail the status of user interface standards, guidelines, development manuals, and formal models.

TABLE 1. Components of usability.

- System Performance
  - Reliability
  - Responsiveness
- System Functions
- User Interfaces
  - Organization
  - I/O Hardware
  - For end-users
  - For other groups
- Reading Materials
  - End-user groups
  - Support groups
- Language Translation
  - Reading materials
  - User interfaces
- Outreach Program
  - End-user training
  - On-line help system
  - Hot-lines
- Ability for customers to modify and extend
- Installation
  - Packaging and unpacking
  - Installing
- Field Maintenance and serviceability
- Advertising
  - Motivating customers to buy
  - Motivating user to use
- Support-group users
  - Marketing people
  - Trainers
  - Operators
  - Maintenance workers

Usability Has Many Aspects

There are many aspects to usability which must be taken into account if a system is to be good (Table 1). Generally designers focus on only one aspect of usability, e.g., knitting pregiven functions together into an user interface. This narrow focus almost guarantees missing the point that usability is made up of many important factors which are often mutually dependent (see Boies, Gould, Levy, Richards, and Schoonard, 1985 for a detailed example).

To illustrate the importance of the various aspects of usability, consider several of those shown in Table 1 that are perhaps least discussed in the literature.

System reliability and responsiveness are the sine qua non elements of usability. If the system is unavailable, it cannot be used. If the system is unreliable, users will avoid it regardless of how good it may be when it works. A survey of 4,448 computer users found that response time was the most important variable affecting their happiness (Rushinek and Rushinek, 1986). With increased system responsiveness user productivity goes up (Doherty and Kelisky, 1979; Doherty and Thadhani, 1982; but see Martin and Corl, 1986). Usability factors go deep, as well as being broad. System reliability and responsiveness go to the heart of system configuration.

It is often alleged that people buy systems for their functions, rather than for their user interface characteristics. (While this is probably historically true, the popularity of MacIntosh reflects the increased value users place on user interfaces.) In the Usability Design Process section below we address how to determine required functions.

Language translation is a serious and time-consuming process. Just the usability detail of putting all the messages in one file, rather than burying them in the code, eases the language translation process -- even makes it possible (see Boies et al., 1985 for a behavioral approach to language translation).

Unpacking and installing computer systems have been greatly improved and speeded up by work of human factors psychologists (Comstock, 1983; Granda, Personal Communication, 1980; Green, 1986).

What we call outreach programs, which include user training, reading materials, on-line help, and hot-lines,

TABLE 2. Usability Design Process -- The Four Principles.

EARLY -- CONTINUAL -- FOCUS ON USERS
Direct contact -- through interviews, observations, surveys, participative design -- to understand cognitive, behavioral, attitudinal, and anthropometric characteristics of users -- and their jobs.

EARLY -- AND CONTINUAL -- USER TESTING
Early on, intended users do real work with simulations and prototypes; their performance and reactions are measured qualitatively and quantitatively.

ITERATIVE DESIGN
System (functions, user interface, help system, reading material, training approach) is modified based upon results of user testing.
Testing cycle is repeated.

INTEGRATED DESIGN
All aspects of usability evolve in parallel;
All aspects of usability under one focus.

often do not reflect the general observation that people learn by doing and by observing others doing.

It is easy to overlook the fact that several other groups, besides the primary users, also need user interfaces, reading materials, and training. These include user trainers, system operators, maintenance workers, and salespeople. Ultimately, usability is seriously affected by the usability of what these people are taught and given. We have observed sales people who avoid giving demonstrations of even simple products (dictating equipment was one example) because of uncertainty about how the products work.

Usability Design Process -- The Four Principles

To design good systems, we believe that you must follow the four principles of system design shown in Table 2 in addressing each aspect of usability. These steps have been developed at IBM Research (Boies et al., 1985; Gould and Lewis, 1983; 1985). They will be discussed in the next main section.

Usability Design Phases

As a chronological framework for discussing what you must do to carry out the four steps in Table 2, we divide the work roughly into four phases: a gearing-up phase, an initial design phase, an iterative development phase, and a system installation phase.

Gearing-Up Phase. This is mainly an information gathering and conceptualization phase. Here you look for good STARTING POINTS, e.g., learn about related systems. You familiarize yourself with existing user interface STANDARDS, GUIDELINES, and any DEVELOPMENT PROCEDURES your organization may have. Each of these resources in capital letters is discussed in separate sections below.

Initial Design Phase. Here you need to make a preliminary specification of the user interface, drawing upon existing and leading systems, standards, and guidelines where appropriate; collect critical information about users and their work; develop testable behavioral goals; and organize the work to be done. Early focus on users takes center stage here. Integrated design, in which all aspects of usability are considered at the outset and evolve together, begins in this phase and is carried into the iterative development phase. All of this is elaborated on below.

Iterative Development Phase. With testable behavioral goals and ready access to user feedback established, continuous evaluation and modification of all aspects of usability can be achieved, as described below.

TABLE 3. Comments we have informally heard from system designers.

"We didn't anticipate THIS."

"But that's not how I thought it worked."

"What do users REALLY want"?

"I'm too pooped to change it now -- it took so long to get here."

"The manual and the user interface are different"!

"Even simple changes are hard."

"Why is user testing so late"?

"Why does user testing take so long"?

"Why don't we do user testing"?

"I would've tested, but...."

"We are surprised that..."

"It worked before..."

"The manual will take care of this..."

"The help system will take care of this..."

"A hot-line will take care of this..."

"We'll take care of it in the NEXT release."

"It's not broken; that's how it's supposed to work."

System Installation Phase. Here concentration centers on techniques for installing the system in customers' locations, introducing it to users, employing the training materials you developed earlier, ascertaining and assuring customer acceptance. (In the case of vendor products, this phase assumes successful marketing has occurred.) The work done on installation, customer support, and system maintenance now receives the ultimate test. For most systems, delivery to the customer should not signal the end of the road, since there are the inevitable follow-ons. Thus, it is just another iteration. If data logging programs to record user performance and acceptance have been incorporated, they will prove useful here. This phase is not discussed further in this paper.

USABILITY DESIGN PROCESS

Beyond Standards, Guidelines, Etc.

Table 3 contains revealing comments made by system designers. Behind them is the realization that relying on a blend of designers' own experiences and following standards, guidelines, or various rational and analytic design philosophies is not sufficient to arrive at good computer systems. Too many systems end up being hard to learn and use, have arbitrary inconsistencies with other systems, and lack the sparkle of insight into what users could really benefit from. From these experiences, several general points shown in Table 4 emerge.

TABLE 4. General observations about system design.

Nobody can get it right the first time.

Development is full of surprises.

Developing user-oriented systems requires living in a sea of changes.

Making contracts to ignore them does not eliminate the need for change.

Designers need good tools.

You can have behavioral design targets, just as you have other capacity and performance targets for other parts of the system, e.g., memory size, calculation time (Gould and Lewis, 1985).

Even if you have made the best system possible, users -- both novices and experienced -- will make mistakes using it.

TABLE 5. Generally required steps in designing good systems.

Define the problem the customer wants solved
Identify tasks users must perform
Learn user capabilities
Learn hardware/software constraints
Set specific usability targets (in behavioral terms)
Set specific usability targets (in behavioral
Sketch out user scenarios
Design and build prototype
Test prototype
Iteratively incorporate changes and test again until:
Behavioral targets are met
A critical deadline is reached
Install system at customer location
Measure customer reaction and acceptance

In response to these problems, several human factors people, apparently working independently, have in the last few years written relatively similar recommendations about how to do better (Bennett, 1984; Bury, 1984; Chapanis and students, e.g., Sullivan and Chapanis, 1983; Damodaran, Simpson, and Wilson, 1980; Meister, 1986; Reitman-Olson, 1985; Rubinstein and Hersh, 1984; Shackel (1984), the Usability Engineering group at DEC, e.g., Good, Spine, Whiteside, and George, 1986; Wixon and Whiteside, 1985; a group at IBM Research, Boies, et al., 1985; Gould and Lewis, 1985. Shackel (1985) has provided a historical summary. All this work represents a coming together of many earlier experiences. For example, the "application of empirical methods to computer-based system design" was the title of an experimental paper more than twenty years ago (Grace, 1966). These principles of system design have much in common with the characteristics of successful businesses, as observed in In Search of Excellence and A Passion for Excellence by Peters and colleagues. These books contain hundreds of examples describing the value of responding to the needs of customers (users).

There are many common procedural steps in the recommendations of all these people, summarized in Table 5, which should be used in carrying out the usability design process.

Principle 1. Early -- And Continual -- Focus On Users

Your Job. Your job is to design a system that has the right functions in it so people can do their work better and like it. You can't figure out what people want, need, can do, and will do without talking with them.

Decide Who the Users Will Be. A first step in designing a system is to decide (a) who the users will be and (b) what they will be doing with the system. This should be done either before starting to design the system, or at a very early stage after obtaining some general ideas about it. Having done this, subsequent decisions, e.g., about system organization, required functions, user interface, must reflect these answers. The user population may ultimately be broader or more heterogeneous. If so, there is no a priori rule for aiming at the "lowest common denominator" or at the "average" user. Rather, your system will have to be tailored and tested for the other groups as well, and design tradeoffs may be required. But establishing one user set is much better than letting this decision continue to slip, as it forces design decisions that otherwise remain open too long. Open decisions about the intended users reflect slipperiness, not flexibility. We believe that the single best way to move advanced technology toward useful applications is by defining an intended user population and what they will do with it.

Designers Often Avoid This. We have observed two serious types of reluctance: one a reluctance to define the users, and the other a reluctance to take the definition seriously. First, as strange as it may seem, designers often avoid coming to grips with the question of who the users will be. This is due in part to the strong (and appropriate) effect the answer will have on subsequent design decisions. For example, designers cannot maintain that they are designing a toolkit system for non-programmers and make it so complicated ("powerful") that even their programming colleagues recognize the contradiction. Reviewers have suggested that this example is a paper tiger -- that no one would believe it. We did not make it up. The reluctance seems to be greatest in research environments, where people's goals are often mixed, e.g., advance the discipline of speech recognition versus build a listening typewriter, advance the discipline of expert systems versus build one to aid human operators or schedulers.

Second, even where designers define early who the users will be, the implications of this decision do not always drive subsequent design decisions in as powerful a way as they should. For example, setting out to design a picture editor for secretaries is laudable, but then building it so that only programmers can use it is inappropriate. We know of one advanced technology interface that is being developed for a specific list of executive users. Yet the designers have never talked with or visited the offices of these executives -- even though they work in the same building as they do. There is little relation between what the designers are building and what the intended users are likely to ever use.

You Can't Rely Upon Descriptive Data. Even if a description of the intended users were as complete as it possibly could be, it would not be an adequate basis for design. It would not substitute for direct interaction and testing.

Methods To Carry Out Early Focus On Users

Several methods for carrying out these principles apply to more than one principle. For convenience, however, we have grouped methods under the principle to which they are probably most relevant.

Talk with Users. Talk with the intended users. Ask them about their problems, difficulties, wishes, and what works well now. You may learn, for example, that morale is very low and that participative design, where some of the workers serve on the design team, is necessary for subsequent acceptance of your system. Tell them -- better, show them if you can -- what you have in mind and get their reactions. They may help you learn about other possible user groups. John Couch, a Ph.D. in Computer Science, worked each weekend incognito in a retail computer store while designing much of the software for the Lisa computer system. "I earned about the fears and frustrations of the average user and the more sophisticated ones firsthand. I believe it was the single most significant source of what we we came up." (Peters and Austin, 1985, p. 9).

You can't expect users to invent radically new ideas. Most users are not trained to think through a design -- but they can comment on your new ideas if you show the ideas in an appropriate form. Their reactions will be particularly helpful to the extent you can make these examples as concrete and relevant as possible. You may learn, for example, that they do not want menus -- they will only avoid them; or that English is a second language for a significant fraction of the intended users, which has serious implications for the design of the reading materials. You may learn how naive you have been about what handicapped people really want. Boies et al. (1985) provide further examples of this method, as well as of other methods mentioned below.

The intended users are the "experts," not, for example, their managers. To illustrate: reviewers of this chapter who were primarily system designers (a main intended user group) provided different recommendations on the topics to be covered, organization, and emphasis than did professional human factors people (more interested in the thoroughness and choice of references, for example) or the Handbook's editors.

Interviewing users focuses the design team's energy on people and not in other directions. You will encounter a dilemma: on the one hand, if you wait for a while to interview users you can ask them more pointed questions; on the other hand, you may be too far down the road to take advantage of other (perhaps more important) insights such interviews provide. The solution is to carry on these discussions throughout the development cycle. Schedule meetings in the users' areas. Put pictures of users, in their own work environments, in designers' offices. Where appropriate, move a designer's office to a user area.

Achieving direct contact with potential users is much better than exclusively relying upon intermediaries, such as marketing research people. Something is lost in the translation. Direct contact helps to reduce the often mentioned problem that designers and users do not speak the same language. Direct contact provides an "insider's view." One Levi executive spends one Saturday a month on the sales floor of a major retailer selling blue jeans. The executive reports that it provides a different view from the market research data. "It was quite an eye-opener for me to sell our own, to watch people buy other people's jeans." (Peters and Austin, 1985, p. 11). Many developers (ironically, rather than users) cannot seem to find time for this, however. We know of one group that was creating an "expert system" to aid computer operators, but members of the group were not planning to interview the operators or watch them work. Visiting directly with hungry or unemployed people has more impact on a person than reading about them. You may not have any idea about what you need to know about users and their environments until you see them. Imagine yourself designing a system to be used by inhabitants of the moon. Certainly you lack familiarity with these users, and need to learn much about them. You will not go wrong if you assume that you lack as much information about the potential users of your own system.

Visit Customer Locations. Visit potential locations for your system, particularly if these environments are new to you. More than a decade ago, Dick Granda (personal communication) visited IBM customer locations and watched them install very large computer systems (IBM 7090 systems). He observed, in the two weeks or so that it took to install such a system, all kinds of unnecessary but tolerated confusion and difficulties. His behavioral observations led IBM to make a number of changes, including color coding the cables, shipping the manuals in the cartons with the boxes they described, and reorganizing the packaging. These changes cut installation time to 1/5 of what it had been.

The Reverend Wyatt Tee Walker was Martin Luther King's trusted lieutenant in the American civil rights marches of the 1960's. "Before King demonstrated to desegregate public housing in Albany, Ga., in December 1961, Walker meticulously examined the area of the planned march, even sending a youngster, a middle-aged man and an elderly woman along the route days in advance to test whether it was manageable." And before the Birmingham, Alabama sit-ins of 1963, Walker "carefully planned the protest, learning as much as he could about the eateries by speaking to blacks in the area." (Waga, 1988.) The point here is that in designing a system to desegregate the South, Walker visited the customer's location and simulated what he intended to do.

Joseph Esherick, recipient of the the American Institute of Architects highest award -- the 1989 Gold Medal -- and famous for the design of San Francisco's Cannery and Monterey Bay Aquarium, says "If you want to design a restaurant talk to waiters, see if they'll let you get back in the kitchen. You'll learn how to ask questions and how to listen." "You ought to pay attention to how things work." (USA Weekend, 1989).

Customers' environments almost always contain surprises for designers. The president of Broadway Stores, a large up-scale retail chain, spends 40% of his time on the sales floor (as told to Stephen Boies, 1989). The 1984 Olympic Message System (OMS) was a voice message system used by Olympians from all countries (Boies, et al., 1985). In a field test of OMS conducted at the desert location of one of the events, it was learned that bugs (the living, multi-leg type) crawled into the computers at night to keep warm. Protection was required or damage could result. If you are designing self-service terminals, you may observe in public places that inexperienced customers are hesitant to insert their credit card for fear of losing it or getting unwanted bills. In factories you may learn that your system must be impervious to grease, dirt, vibration, and caustic materials. In schools you may learn how naive you are about today's learning problems and achievements. In offices you may learn about the sociology that affects the use of a system. A large insurance company that wanted its thousands of quasi-independent field agents to purchase and use terminals learned that the agents would use them only if they were quiet and did not generate much heat in their typically small offices.

Observe Users Working. Visit the workplaces of users. Most systems are follow-ons to existing ones. Learn what is hard about the existing work so you don't make the same mistakes. Observe potential users doing their jobs. You may feel self-conscious at first. Hang out. Blend in with the woodwork. Ask non-threatening questions as workers become less self-conscious. It will all go well. You may learn that operators avoid changing tapes and ribbons or adding paper to machines; that secretaries avoid transferring calls for fear of making an error. You may learn that space is very limited, or that workstations must be shared, or that no one comes to training sessions, or that no one reads manuals. You may learn what users do when they don't know something or get in trouble, which should help you in designing your outreach program. See what they have difficulty with and what they dislike. You may learn that construction workers don't wear their reading glasses and don't want to press small keys, particularly while wearing heavy gloves; that office workers will not use an electronic mail system that automatically recognizes their speech if they have to wait thirty seconds to activate it before each usage; that the operators expect that the new system will respond much faster than the present one (had you planned to make it that way?); that airline passengers lose their credit cards by inserting them in the wrong place (ticket dispenser); that management has no intention of buying any manuals. The implications of these findings must then be incorporated into the design of your system.

Videotape Users Working. Make a videotape of a few users working, and show it to other members of the design team or management. Brief (2 min.) videotapes of users having difficulty and illustrations of home-made remedies to solve design inadequacies carry more punch with people who count than do tables with lots of numbers. The tables, incidentally, can be obtained only with much more effort.

Learn about the Work Organization. Here the emphasis is upon understanding the organization of the work that your system is intended to help. This is particularly important where many different user groups are involved (see Mumford, 1981). A system with an user interface that was adequate and well liked, as reported in one case study of an IBM hospital care system (Fitter, Brownbridge, Garber, and Herzmark; 1985), was nevertheless under-utilized in its two installations because there was a mismatch between the system organization and the hospital organization.

Thinking Aloud. Have potential users think aloud as they are doing their actual work (see Lewis, 1982). Doing this while actually performing their job may yield different insights than having them reflect on their work later. There will be real time comments that may provide details important for all the little decisions you make everyday in designing your system. Like a politician who visits his constituency, you will learn what the real issues are. You may learn that the sales clerks don't want to fold clothes, check ids, have long lines at their check-out counters, or make change because of the implications of possible mistakes; that bank tellers don't want to share workstations or have it known that they can't work with fractions; that warehouse workers will probably avoid using a workstation if they have to walk a long way or wait for it to warm up each time they need to use it.

Try It Yourself. Sometimes it can be rewarding to try a worker's job yourself. You may learn, for example, that glare on the screen is really bothersome, a slow system leads to rude customers, your arms get tired, you can't handle all the interruptions. Of course, you must be careful not to confuse your expertise with that of the intended users. We have been told that Kentucky Fried Chicken executives are required to work for a period in retail outlets.

Participative Design. Make intended users part of the design team. This is particularly effective when designing an "in-house" system which you will be responsible for introducing once it is completed, e.g., a quality control system for your factory, a data entry system for your distribution center, an automated library for your division. Mumford (e.g., 1981) has written several papers on participative design. Participative design is not just a good thing to do politically. Experienced workers know a lot more about what they do than do drop-in designers. (Don't you know more about your work than, for example, a consultant to your manager's manager?) Glasson (1985) describes various roles users can play here. Eason (1982) describes some difficulties in accomplishing participative design.

Worker involvement in design allows workers to participate in decision-making about conditions that affect their work lives. Besides leading to better systems, this process leads to an increase in pride, self-esteem, and perhaps autonomy. Maybe productivity too. People want to be actively involved in their work, and in the decisions that affect their work lives. Sometimes people have to be helped here, though.

Expert on Design Team. Using an expert to consult on the design team of OMS, in that case a recent ex-Olympian, was valuable in designing that voice message system (Boies et al., 1985). This person was a consultant and not a full-fledged member of the design team. In some cases, for example having a physician as part of the team to design a patient monitoring system, might be more valuable if the person is committed to seeing the project to completion and not simply a consultant. Putting experts on the design team is simply another way to phrase "participative design." The term is sometimes used when the intended users are highly skilled individuals. Keep in mind, however, that people who do lower skill jobs are nevertheless the experts at their jobs.

Task Analysis. Probably the most formally worked out method of learning about users' jobs is task analysis. Task analysis is an analytical process used to determine the specific behaviors required of people in a man-machine system (Air Force definition). Formal task analysis is usually "operator-oriented," rather than "discretionary-user-oriented." This reflects the nature of most jobs in which it is used. Task analysis, says Drury (1983) in a tutorial, is a comparison between the demands the task places on the human operator and the capabilities of the operator to deal with them. It is the process of identifying and reporting the significant work activities, requirements, and technical and environmental conditions. It is usually carried out through observation, interviews, or questionnaires. A formal task or job description results.

Formal task analysis is regularly conducted by the Armed Services and government, and their contractors. This formality reflects the division of labor that often occurs in large system projects, e.g., a new, multi-person space system or a defensive missile system. Descriptions of pieces of a proposed system, e.g., the "personnel subsystem," are collected by one group and passed to another group, who passes them to yet another group. The literature on task analysis of government-funded systems (e.g., Kloster and Tischer, 1985; Meister, 1986) and the literature from the civilian computer industry (as illustrated, for example, in the CHI proceedings) have developed independently, it appears. There are many variations in task analysis. Mentemerlo and Eddowes (1978) reviewed many, and concluded that it is not possible to have a single cookbook approach that is universally applicable. While formal task analysis is valuable, the division of labor noted above can provide layers of insulation between the ultimate users and various designers along the way. It should not be carried out without having designers talk directly with users.

There are many informal, valuable informal task analysis approaches. At IBM Boca Raton, Happ and human factors colleagues (personal communication) have designers and users write each step that they believe is necessary to complete a task on a separate slip and put it on a wall. Happ reports that the design (with its flaws) comes to life, and that they discover multiple steps, confusions, left out operations, and exceptions. What results is an improved early conception of what will be required. In a different approach, not aimed at system design but aimed at identifying what executives do, Mintzberg (1976) followed each of several executives for two weeks, carefully observing and measuring their daily activities and talking with them about them.

Surveys and Questionnaires. The data obtained from surveys and questionnaires can be useful. Talking with a group of users first, and with other members of the design team, is necessary to know what questions to ask. Sometimes answers to questions can seem sterile, e.g., average years education, employee turnover. But the implications of these answers are powerful. If the users have relatively little education and high job turnover, then training must be brief and inexpensive.

Testable Behavioral Target Goals. Most new systems specify in advance physical performance and capacity targets, e.g., memory swap time, MIPS, mean time to failure estimates, instruction times. Explicit behavioral targets that new systems must meet can also be established (e.g., Bennett, 1984; Carroll and Rosson, 1984; Gould and Lewis, 1985; Shackel, 1985; Wixon and Whiteside, 1985). An example target goal is: business professionals with no experience using a new office services system must be able to retrieve and read three brief electronic messages from their system mailbox and reply to each within twelve minutes. They can ask the experimenter (a simulated "hot-line") for help no more than once. These explicit behavioral benchmarks go beyond general expectations, often implicit only, that a new system should not require more time or lead to more errors than the old (manual?) Table 6 provides additional partial examples of testable behavioral specifications.

Measurable behavioral targets, and where your developing system stands with respect to them, give management a metric to understand what progress has been made, and what is still required. This makes it possible to judge usability on the same basis as other system components.

Behavioral targets give phrases like "user friendly" or "easy to use" a technical basis. Wixon and Whiteside (1985) have worked out a methodology called "usability engineering" in which specific behavioral goals play a key role. The time requirements, the number of errors and help attempts, and the user acceptance ratings that form the behavioral targets are arrived at by discussions with users, discussions among the design team, and the characteristics of the existing systems that will be improved upon. Measurable usability targets stated in behavioral terms are required to determine whether the finished product fulfills its usability goals (Good, Spine, Whiteside, and George; 1986)

TABLE 6. Additional examples of testable behavioral specifications.

Example 1. Twenty experimental participants, familiar with the IBM PC but unfamiliar with query languages, will receive sixty minutes training using the new on-line query training system for novice users. They will then perform nine experimental tasks.

On Task 1, 85% of these tested users must complete it successfully in less than 15 minutes, with no help from the experimenter. They may use all reference and help materials, but no hot-lines. Task 1 consists of six steps:

1. Create a query on the displayed query panel using the table SCHOOL.COURSES.
2. Delete all column names except COURSE and TITLE.
3. Save the current query with the name CTITLE.
4. Run it once.
5. Get the current query panel displayed.
6. Clear the current query panel so that it contains nothing.

Example 2. For this integrated editor, any displayed messages encountered in the process of creating documents, tables, or drawings will be understood by secretaries with one or more years of word processing experience, as demonstrated by the ability of at least 90% of them to paraphrase correctly these messages. The system will provide error recovery mechanisms suitable for these users, as evidenced by the ability of at least 85% of them to execute successful recovery strategies in seven of the attached eight scenarios (adapted from Carroll and Rosson, 1984).

Example 3. Nine of ten participants, whose first language is not English, must be able to walk up and use this vending machine, get the target item within twenty seconds, and get the correct change. This must be done without watching anyone else use it and without asking the experimenter any questions.

Example 4. Test participants, a random sample from customers in retail computer stores who are willing to participate in this experiment, must be able to unpack this prototype home computer and connect the display, printer, mouse, keyboard, and special-effects card. They must initiate the system and then set the date and time on the computer. This work must be completed by at least 17 of the 20 participants in 60 minutes, using only the instructions contained in the manuals in the boxes.

A Checklist. Table 7 is a checklist to help you carry out early -- and continual -- focus on users.

TABLE 7. Checklist for achieving Early -- and Continual -- Focus on Users.

____We defined a major group of potential users.

____We talked with these users about the good and bad points of their present job (and system if appropriate).

____Our preliminary system design discussions always kept in mind the characteristics of these users.

____We watched these users doing their present jobs.

____We asked them to think aloud as they worked.

____We tried their jobs (where appropriate).

____We did a formal task analysis.

____We developed testable behavioral target goals for our proposed system.

Principle 2. Early -- And Continual -- User Testing

Your job is to design a system that works and has the right functions so that users can do the right things. You won't know whether it is working right until you start testing it.

From the very beginning of the development process, and throughout it, intended users should carry out real work using early versions of training materials and manuals, and simulations and prototypes of user interfaces, help systems, and so forth. The emphasis here is upon measurement, informal and formal. The basic premise is that you cannot get it right the first time, no matter how experienced or smart you are. This observation is not limited just to computer system designers. Heckel (1984), in writing about software design, asks "If Ernest Hemingway, James Michener, Neil Simon, Frank Lloyd Wright, and Pablo Picasso could not get it right the first time, what makes you think you will?" Heckel quotes others: "Plan to throw one away" (Fred Brooks). Rewrite and revise...it is no sign of weakness or defeat that your manuscript ends up in need of major surgery" (William Strunk and E. B. White). "The two most important tools an architect has are the eraser in the drawing room and the sledge hammer on the construction site" (Frank Lloyd Wright). If you measure and then make appropriate changes you can hill-climb toward an increasingly better system.

Methods To Carry Out Early -- And Continual -- User Testing

Printed or Video Scenarios. As a starting point, sketch out a few user scenarios on paper and show them to members of the design team. Provide exact details, e.g., the exact layout and wording on the screen, exactly what keys users must press, and the response of the system. These details will of course change -- but this presumptive exactness stimulates the right level of discussion. Members of the design team now must react to a proposal; removed is the misleading comfort of not realizing the design conflicts that lie hidden when these details are missing. These scenarios do not just lead to arguments about surface characteristics of an user interface. Since they inherently specify functions, it is our experience that they lead to discussions about required functions, and how they should be organized. This in turn will quickly identify deep systems organization issues.

Then carry the process one step further. Type up these scenarios, or perhaps make a brief video demonstration of them if you are designing a system where graphics, animation, or color is dominant. They are now in shape to be shown to prospective users for their reaction. In the meantime, other members of the design team have developed some shared realization of what the entire project is trying to create.

What you have done with this procedure is to identify and organize functions in a way that intended users can understand and react to. You are quickly getting informal data before even writing a line of code. You are designing the system from the users' point of view.

Early User Manuals. Begin writing the user manual before any code is written. Intended users can react to this helpfully, since the system is being described in the appropriate fashion. If you get inappropriate reactions, start re-writing the manual. When people ask questions, you have new items for the manual. The printed and graphic scenarios, prepared as suggested above, can be included as examples. This works. Designers at Digital Equipment Corporation wrote the user manual simultaneously with beginning to design an user interface for a workstation. They found, from informal evaluation of their user manual, that they were making their system too complicated, and modified it while there was still a chance to do so (Rubinstein and Hersh, 1984). Of particular significance, they also found that the user manual became their definitive design document -- in spite of the fact that all the traditional documents were also available. Cowlishaw (1984) first wrote and circulated for review the documentation for each major section of the the successful REXX language before he implemented that section. (REXX is an IBM interpretive language often used for command and macro programming, prototyping, education, and personal programming.) He reports that "The writing of documentation was found to be the most effective way of spotting inconsistencies, ambiguities, or incompleteness in a design. The majority of usability problems were discovered before they became embedded in the language." In making OMS, designers at IBM Research wrote the user guide first (Boies et al., 1985). Based upon people's reactions to it, they made changes to the planned system functions and organization. The designers put immediately into the user guide details that people suggested ought ultimately to be there.

With this approach,early manuals continually evolve, with their final version being very different from their first versions. In contrast, writing manuals today is usually done sometime after the system has been designed and implemented -- and by people in a different department. In just this situation, an IBM manual writer took the initiative to write part of a manual early. In explaining how to turn on the system, the writer made up a procedure, since the designers had not yet settled on one. A hardware designer read the manual, liked the "solution," and adopted it. The point: manuals can influence what other designers do. An experimental case study of re-writing an existing text editor manual provides several good leads on how to go about writing a user manual (Sullivan and Chapanis, 1983).

Mock-ups. Seminara (1985) discusses the advantages of mock-ups and models in developing systems, and draws on a power plant example. Boies et al., (1985) used a wooden mock-up to begin development of the kiosk that became an essential part of their outreach program for OMS. They put this mock-up in the main hallway of the Research Center. It led to innovative solutions to a difficult training situation they were facing. Rather than slow down the development process, it contributed to getting things done on time in a six month development schedule. The comments of passers-by were beneficial in designing all parts of the kiosk, e.g., physical locations of pamphlet holders, display, and telephone; messages; display color and scrolling techniques; security. It also led to the identification of issues that might not have been otherwise as easily envisioned, e.g., air conditioning requirements, required room inside for maintenance. As the life size mock-up evolved, it became the design model that was replicated by the fabricators.

Simulations. Much informal experimentation can be carried out by simulating important parts of the system. Sometimes this can be done with paper and pencil (Thomas and Gould, 1975). Sometimes successful simulations can be carried out with computer systems, separate from those that will ultimately be used in a product (Gould, Conti, and Hovanyecz, 1983; Kelley, 1984). Code iteratively developed in simulations need not be thrown away, but can be used in the actual product if planned properly (Richards et al., 1985). The OMS case history gives a detailed example of simulation (Boies et al., 1985).

More formal experimentation (i.e., carefully designed and controlled laboratory experiments with serious statistical analysis of the results) can be carried out with simulations also, if you have the expertise to do this. Erdmann and Neal (1971) simulated an airlines reservation system in an airport by having a hidden reservation agent actually respond to customers who were using it. Gould, Conti, Hovanyecz (1983) simulated a listening typewriter by having a hidden typist actually enter what users said (Figure 1). Kelley (1984) put himself in the human-computer loop to answer questions his natural language calendar system could not. Good, Whiteside, Wixon, and Jones (1984) intercepted inappropriate commands of novices using an electronic mail system, substituting the correct ones, but adding the inappropriate ones to a new extended command set. Some system tools allow the creation of relatively complete early collections of on-line user interface frames connected only to the few functions so far written. All these simulation techniques identified both how people used the systems and and how they felt about them.

Early Prototyping. Early prototyping can be made possible through the use of designer toolkits or user interface management systems (see section on Software Tools below). Try to develop pieces of your system to the point where potential users can carry out pre-defined problems (even if other pieces of the system are not yet working). You will learn things you have possibly missed through simulations, e.g., effects of multiple simultaneous users, side-effects of functions. Rapid prototyping changes work organization. It allows and stimulates discussions among workers. The discussions are much different than without it. People that could not previously make a contribution because of the form that the work existed in can now contribute usefully.

You can measure people's performance and feelings. In at least some cases how people feel about a new system better predicts actual discretionary use of it than does how effectively these people actually use it (Davis and Reitman-Olson, 1986). There are many testimonies to the value of early prototyping. For example, Woodmansee (1985) who, in writing about the experience of developing Visicorp's Visi On (R), suggests it could have been even better had early user interface prototyping been done. Alavi (1984) found that prototyping, in contrast to the traditional life-cycle approach to system development, facilitates communication between users and designers, but is harder to manage and control. Mitch Kapor, the inventor of Lotus 123, in reflecting on its development process, indicated that they continually prototyped their work. He served as the main experimental user, sometimes trying to get real work done, other times demonstrating the prototype to others and getting their comments, other times simply trying out a new version and providing feedback (talk given at the IBM Research Center, Hawthorne, 1989). See Case Study Evaluations below for examples of early prototyping.

Prototyping is expensive, but necessary. Our a priori understanding of users is imperfect at best. Guidelines and generic behavioral research (see Gruenenfelder and Whitten, 1985) provide at best only approximately good recommendations. And our systems contain bugs and inconsistencies.

Sometimes system programmers or management simply do not believe reports that users are having a difficult time learning, for example, a new prototype system. Videotapes of users having problems are often much more convincing than charts and oral reports. (Where possible, the effect can be even more powerful if you can get them to watch early testing themselves.)

Early Demonstrations. Demonstrate working pieces of your system to anyone who will take the time to watch. Simply going through the motions of using it, and honestly observing others' reactions will be instructive. Of course, it is even better to let them try it themselves on a brief task. Even a simple task can be revealing. For example, when the touchscreen does not work, you may be surprised to learn that users do not lift their finger and press again as your algorithm requires, but rather just press their finger even harder and wider. This problem would not show up in your demonstrations, but only when somebody else tries it. Successful demonstrations of pieces of your system and manuals give management and customers confidence that you are making progress.

Thinking Aloud. Performance measures, such as time and errors, do not give a clear indication of what is bothering users or what may be the source of an user error (Lewis, 1982). People simply do not make the mental transitions ("leaps") that system or manual designers expect of them. At Carnegie-Mellon University's Communications Design Center, students and faculty test out new user manuals by studying individuals talking out loud as they try to get things done. This results in both gross changes to the manual, e.g., identification of missing sections, and subtle modifications. Human factors people at DEC found that having two users think aloud, i.e., basically talk to each other, as they tried to install a small computer made it easier to obtain verbal n protocols from some users and led to additional insights (Comstock, 1983; see also Rubinstein and Hersh, 1984). Of course, requiring users to think aloud can affect the precision of time and error performance measures, and so when precise performance measurements are sought, then you may not want to collect verbal protocols.

Make Videotapes. Besides being useful for measuring time, errors, and user attitudes, brief videotapes of users attempting to use a new system have tremendous impact upon management, especially where users are having problems (see also Rubinstein and Hersh, 1984). Designers often find it impossible to believe that others cannot use their system. These vignettes can be mind-opening. These are often viewed over and over, and elicit vigorous reactions -- from colleagues and particularly from management.

Hallway and Storefront Methodology. Gould et al (1987) coined this term on the basis of their experience of putting parts of OMS during its development in the hallway of the IBM Research Center in Yorktown. By putting a mock-up, simulation, or an early prototype in an obvious public place (or restricted access "public" place in the case of a company confidential system), passers-by just naturally are attracted to use it. This provides a source of invaluable comments and surprises. Tognazzini in his talk at CHI86 described how Apple Computer started doing usability testing at local computer stores, "dragging along the programmers," as he put it, "and having a bunch of random people go through their latest software." "It was very effective," he said, "because not only did they quickly discover the pitfalls, but the programmers saw them too, avoiding all those arguments."

In hallway and storefront methodology, you must react quickly to all good suggestions, however, if you want to keep the flow of comments coming. This demands good tools for interface designers (see below under Iterative Design). Even trivial errors must be quickly corrected. For example, if a person notices a misspelling, correct it quickly. If it is there the next day, it will begin to affect the confidence of the passers-by that you are taking their comments seriously. Making changes quickly requires the project to be organized for integrated usability design (see above). What gets learned in storefront and hallway methodology is valuable for user guides, user interfaces, display sizes and colors, identification of required functions, help systems, and the design and looks of the workstation.

As an added benefit, hallway and storefront adds zest, gusto, fun to a project. It can give a project momentum that was not previously there. People notice what you are doing. Members of the project see better how their work fits in, and what other members are doing. Your project begins to emerge from a sea of other projects that may be going on around you.

Putting a mock-up or prototype in the hallway can help your cause with system colleagues whom you may be having trouble in getting to work on aspects of the system that contribute to usability. Passersby's reactions will quickly identify what is important, and may very quickly lead to a work re-organization. The whole project will benefit, for the right working relations will be identified.

Some people have expressed a concern that they cannot protect proprietary systems with hallway and storefront methodology. Maybe so in some cases, but there is still much that can be learned. At IBM Boca Raton lab where IBM PC plans are carefully guarded, the human factors people have placed displays in the library and cafeteria and obtained reactions from passers-by about color and screen layout. They were able to do this without associating these designs with any particular future product.

Computer Bulletin Boards, Forums, Networks, and Conferencing. Existing, extensive computer networks allow designers to send out a partial or an entire new user interface and obtain feedback from users all over the world -- most of whom would otherwise be unknown to the designer. Cowlishaw (1984) did this in a very serious way during the four years he was developing REXX, an interpretive language often used for command and macro programming, prototyping, education, and personal programming. "The most important factor in the development of REXX," he writes, "began to take effect when the first interpreter was distributed over the the IBM communications network known as VNET. (This network links over 1400 mainframe computers in forty countries.) From the beginning, many hundreds of people were using the language...from temporary staff to professional programmers. (They) provided immediate feedback to the designer on their preferences, needs, and suggestions for change. An informal language committee then appeared spontaneously and communicated among themselves and with the designer entirely electronically. The discussions...grew to hundreds of thousands of lines. Using the network, the designer could interactively explain and discuss the changes that were required." Cowlishaw concludes modestly that "Many if not most of the good ideas embodied in the language came directly from users. It is impossible to overestimate the value of the direct feedback from users." One factor that motivated users to provide feedback was the speed with which Cowlishaw responded to them.

Electronic bulletin boards allow you to "tack up" electronic requests for help, advice, comments. Like traditional bulletin boards, for example in universities or in super markets, people unknown to you will read your message. Computer conferencing facilities have enhanced the basic bulletin board notion by giving the reader many aiding functions. For example, the reader is provided with ways to look up all references on your topic, all messages you may have posted in the past on this subject, an easy way to reply to your request, and ways to run your program should you post a program on the bulletin board or forum (Flavin, Williford, and

Barzilai, 1986). Oftentimes forums develop on the basis of user comments about using an already-designed system, e.g., Lotus 1-2-3. These forums center on users sharing tricks in using the system of interest, and feedback to the designers for the next version. Clearly, however, on-going forums, either public or private, can be useful while design of a new system is on-going. People's comments are usually informal, but informed -- and come from ego-involved users.

Formal Prototype Test. Much of the emphasis so far has been upon informal experimental results, e.g., create a scenario, simulation, or prototype; measure the performance of some users doing real work; get their reactions; modify the user interface; then repeat the process. Informal empirical and experimental work are very valuable, and give an idea of where you stand vis a vis the behavioral targets you established earlier.

TABLE 8. The percentage of participants who successfully completed all three programming problems in each of four iterative tests of a system under development. Data adopted from Bury (1985).

Iteration 3	75% tasks completed
Iteration 4	33% tasks completed
Iteration 5	65% tasks completed
Iteration 6	92% tasks completed

NOTE: Data are based upon the three problems common to all iterations. Participants' performance improved during the first three iterations. The results "go up and down" because prior to Iteration 4 a new training approach was developed, and the deleterious effects of it were identified.

These activities can be carried out without some of the technical skills of trained human factors people, e.g., psychophysics, experimental design, or statistics. But we certainly encourage formal experimentation where possible. If skilled human factors people are available to develop the experimental and statistical designs, then an even more accurate and valuable assessment can be made. As shown in Table 8, Bury (1985) provides an example of how formal experimental iteration can improve a system. "Give me numbers like iterative design provides," said one corporate officer, "and you take user interfaces out of the realm of taste and preference only."

There are many formal behavioral methods for designing software systems (see, for example, a National Academy of Science/National Research Council recent report which summarizes many (Anderson and Olson; 1985)).

As your project nears its end, you are really busy and of necessity have become part salesperson to push it to completion. Thus, even though you have done much behavioral work already, it is better if you can have an outside group do the final evaluation. It is just too hard to be objective under the final pressures, even though you have been objective up to this point.

Try-to-Destroy-It Contests. The design team of OMS, near the end of development, turned over their system to a group of college students and let them try to find bugs, crash it, break into it, etc. (Gould, et al., 1987). College students require little extrinsic reward for doing this type of work! A sociology develops that may initially be humbling to you, but is ultimately of immense value. If there is a need to keep this test proprietary, it is still not hard to find cooperative, motivated people to carry it out.

Field Studies. Laboratory and hallway studies go only so far. Each methodology mentioned so far yields somewhat different information. Putting your system into the field for a test reminds you of problems that that you have put out of mind or identifies problems that other methodologies do not get at (see Boies et al. (1985) for a detailed example). While laboratory studies of computer installation are valuable, relates John Whiteside of DEC (personal communication, 1987), it took a field study to identify the customer problem of what to do with all the cartons and packing materials after installing a small system on the 32nd floor of a downtown building. Field test clearly suggest priorities about which problems to solve first. So-called "early customer shipment," vendor-customer agreements, and early site installations are mechanisms for doing field tests.

TABLE 9. Checklist for achieving Early User Testing.

_____ We made informal, preliminary sketches of a few user scenarios -- specifying exactly what the user and system messages will be -- and showed them to a few prospective users.

_____ We have begun writing the user manual, and it is guiding the development process.

_____ We have used simulations to try out the functions and organization of the user interface.

_____ We have used mock-ups to try out the functions and organization of the user interface.

_____ We have done early demonstrations.

_____ We invited as many people as possible to comment on on-going instantiations of all usability components.

_____ We had prospective users think aloud as they used simulations, mock-ups, and prototypes.

_____ We used hallway and storefront methods.

_____ We used computer conferencing forums to get feedback on usability.

_____ We did formal prototype user testing.

_____ We compared our results to established behavioral target goals.

_____ We met our behavioral benchmark targets.

_____ We let motivated people try to find bugs in our systems.

_____ We did field studies.

_____ We included data logging programs in our system.

_____ We did follow-up studies on people who are now using the system we made.

Follow-Up Studies. Once a system has been released, studying how actual customers use it has value for subsequent releases and related new products. This work serves as a validation of the earlier prototyping and iterative design efforts, and it is particularly important in assessing the usefulness of various functions and what new functions are required. These studies illustrate the need to provide automatic data collection tools in your system.

Studying hot-line calls or service center requests and complaints related to your system can be insightful in identifying problems users are having. For example, Sony Corporation's telephone-based "customer-information center" receives 1200 phone calls per day. A computer record summarizing the questions people asked is used to modify the company's user manuals (Schrage, 1986). At Apple, all senior executives learn of real user problems by listening to them on a toll-free 800 customer call-in line. Occasionally they try to answer the question (Peters and Austin, 1985).

A Checklist. Table 9 is a checklist to help you carry out early -- and continual -- user testing.

Principle 3. Iterative Design

The key requirements for iterative design are:

Identification of required changes.

An ability to make the changes.

A willingness to make changes.

The required or recommended changes can be identified with measurements made on intended users and the results compared against previously established behavioral goals. To make these changes, however, requires that designers have good tools, and that the work is organized in a way that enables them to be responsive.

When you find a problem, what to do about it may not be clear. There is no principled method to determine what the solution is. There are only empirical methods -- to be used after careful analysis, critical thinking, and innovation have been applied. The empirical methods can either be used during system development or they, in effect, will be used after the system is delivered -- which is usually an inopportune time. Sometimes you may make changes that will cause other problems. You will only know if you test them.

Methods To Carry Out Iterative Design

Software Tools. There has been recent widespread recognition of the importance of providing designers with good tools. In the past, particularly as deadlines approached, there has been great reluctance to make even simple changes to an user interface because these changes might introduce bugs into the application code. That is, code for an user interface and an application were mixed together. Experience with this problem has led to the development of toolkits (e.g, the MAC toolkit) and user interface management systems (UIMS).

You need good tools. First, tools are needed to make changes in an user interface possible and easy. Without this, there can be no iterative design, which is so necessary for the design of good systems. These tools can be the basis of rapid prototyping. Second, they bring an user interface within the control of human factors people and other non-programming specialists who understand the application. Third, good tools reduce cost by making individual programmers and designers more productive, and by speeding up schedules. This in turn further reduces cost by reducing disproportionately more the number of required people. Fourth, inherent in tools are "ways of doing things." These can automatically contribute to improved usability (or reduced usability if they enforce bad constraints). Fifth, good tools can facilitate user interface consistency and cross-system consistency.

The basic characteristic of UIMSs is that there is a separation between an user interface and the functions that it uses. That is, there is a well defined interface between that portion of the code that contains an user interface and that portion of the code that implements the functions of that user interface. The driving force for this separation is that it enables an user interface to co-evolve with an application. This separation reduces or eliminates the need to test application code each time a change is made in an user interface. That is, an user interface can be changed independently from the function code.

UIMSs are evolving from several orientations. Some UIMSs are created in the service of developing a specific system, e.g., as was the case in IBM's voice messaging systems (Richards, et al., 1985). These may have generality to other applications in the same domain. Some UIMSs are available only within the company that developed them, e.g., Xerox's Trillium (Henderson, 1986); tools from several aerospace companies (Overmyer, 1987). Some UIMSs provide a means of rapid prototyping, but once an user interface is settled on, then the system must be re-coded. Softright's DEMO is an example. The MAC toolkit, designed for application programmers, aims at reducing development costs and encouraging consistency in an user interface across applications. Some UIMSs are commercially available, e.g., Apollo Computer's ADM: Cosmic's (NASA software) TAE. The intent here is to handle a broad range of applications. Good general references to read on UIMSs are Bennett (1986); Buxton, Lamb, Sherman, and Smith (1983); Green (1985); Pfaff (1985).

Designers who have actually used UIMSs in the development of their own systems describe them in very positive terms (Richards, et al., 1985; Shulert, Rogers, and Hamilton, 1985; Hayes, Szekely, and Lerner, 1985)

Are UIMSs and toolkits easy for non-programmers to use? They are evolving, but ease of use does not seem to be the prime consideration. To be successful in using an UIMS, you must at present become a skilled user of it, even if not a skilled programmer.

System Development Work Organization. Required changes to various aspects of usability (Table 1) cannot be carried out in a coordinated way if the work is not organized to make change possible. Indeed, none of this will work at all if there is not a willingness to live in a sea of changes, and react to required changes quickly and appropriately. Busy white collar workers live in a sea of interruptions, and realize that unexpected changes are part of their job. It is part of the job of designing good systems to manage change, and not make contracts to ignore required change. You can't manage change by pretending it is not needed.

A Checklist. Table 10 is a checklist to help you carry out iterative design.

TABLE 10. Checklist for carrying out Iterative Design.

____	All aspects of usability could be easily changed, i.e., we had good tools.
____	We regularly changed our system, manuals, etc., based upon testing results with prospective users.

Principle 4. Integrated Design

As explained in Boies et al. (1985), we recommend that all aspects of usability evolve in parallel. At the outset, work should begin on sketching an user interface, user guides, other reading materials, the language translation approach, the help system, and so forth. In order for this to happen successfully, all aspects of usability should be under one focus or person. Of course, that person will probably have other usability people working for him/her. Usability cannot be coordinated otherwise. This one-focus recommendation presumes line-management responsibility, and is thus different from Usability Committees (Demers, 1984).

An example illustrates the need for integrated design. During a field test of OMS (Boies, et al., 1985) before the Olympics started it was learned that the word "Olympics" had to be changed to "Olympic." This would seem to be a trivial change. But alas. All user interface messages and all help messages with the word "Olympics" had to be changed. Since these messages were recorded in twelve languages, speakers for each of these languages had to be obtained. The user guides (and other printed material) had to be changed. These were in twelve languages also, some of which involved alphabets that most printers do not handle, e.g., Arabic, Russian. The lettering on signs had to be changed. Since one person was in charge of usability (and OMS was organized so that all messages were in one file), this clearly modest change was done in all appropriate places. Under the usual organization, however, this would have been much more difficult. For example, the required change might have been recommended by human factors people, who would have had to negotiate it with the several different managers (e.g., the project manager, user interface manager, help system manager, documentation manager, advertising manager). Even if they were all convinced, they would then have to negotiate it with their programmers or other relevant staff people. Then there would be the required inquiries to be sure all relevant people agreed to do it, and did it. This trivial change is typical of many, many trivial changes one discovers that need to be made during the last stages of development. He who has the stuff in his computer has the power. A more detailed example of the need for integrated design is in Boies, et al. (1985).

Methods To Carry Out Integrated Design

A project can be managed to only a few goals, e.g., low cost, processing speed, compatibility with the past, reliability, short development schedule, usability. With the methods described in this paper, you can measure usability, therefore control it, and therefore manage it. Integrated design is an essential approach if one of the goals of your project is usability.

The methods just outlined under early focus on users and those mentioned below to carry out user testing must be brought to bear on all aspects of usability. Technically, these methods are sufficient to guarantee an acceptable system. The main difficulty in carrying out integrated design will be organizational. Integrated design requires a departure from fractionated development practices where various aspects of usability are developed in different loosely-related departments, divisions, cities, companies.

Integrated design assumes a recognition at the very outset that usability is important, that it includes many factors (Table 1), and that work must begin on it from the start. Integrated design requires a sincere dedication to manage for usability.

Integrated design requires that one group, at the very beginning, be given sufficient resources (money, personnel, time, authority) to drive and control usability, and to invent what is needed to make usability good. This organization may have critical mass early enough to be an effective lobby for usability -- to assure that usability gets its share of the resources of the project. Integrated design requires that this group sign up to guarantee good usability. Their duties include carrying out the methods described under each of the other three principles, or see that others do.

Today development groups are not organized to facilitate integrated design. Development of the functions, user interface, manuals, help system, training materials, etc. are often each done in a separate department in large projects.

Because of these traditions integrated Design may be tough to carry out in many organizations. It requires that the usability people be outstanding, be given the responsibility (and accountability), and have good tools. It is not just a plug for more jobs for human factors people. The responsibility will be extremely demanding, especially on large systems. Very special people will be required. We have been told that no one person could possibly control all aspects of usability on large systems. This is simply not logical, since there is generally one person in charge of the whole system (of which usability is only a part).

A Checklist. Table 11 is a checklist to help you carry out integrated design.

TABLE 11. Checklist for achieving Integrated Design.

____We considered all aspects of usability in our initial design.

____One person (with an appropriate number of people) had responsibility for all aspects of usability.

____User manual

____Manuals for subsidiary groups, e.g., operators, trainers, etc.

____Identification of required functions

____User interface

____Assure adequate system reliability and responsiveness

____Outreach program, e.g., help system, training materials, hotlines, videotapes, etc.

____Installation

____Customization

____Field Maintenance

____Support-group users

Evaluation Of The Usability Design Process

In recent informal surveys we have made of system designers, we have been struck that most of their technical concerns would be addressed by using the methods just described.

A distinguishing feature of successful American companies e.g., fast food vendor, grocery store, computer manufacturer, and government units is their long term dedication to customer satisfaction (Peters and Waterman, 1982; Peters and Austin, 1985). Methodologically, people in these success stories stay close to their customers ("users"); they wander around. We have been told that several years ago a large corporation studied their twelve major business failures. They concluded that in each failure they did not know that particular "business" or "application" or "user set" for which they were creating a new product. IBM's market success has often been attributed to "knowing the customers' business better than they." The point here is that these examples illustrate the value of knowing and serving users.

We have talked about the process of design. As part of this process, it is possible to select as starting points good designs from an existing inventory, e.g., pull-down menus, pointing and selecting with a mouse, icons, Lotus "menu-bar," multiple windows on a screen. These designs have themselves stood the test of empirical usage. (see section on Starting Points below)

Comparison to Other Approaches. Gould and Lewis (1985) have compared usability design with other design approaches, e.g., getting it right the first time. You simply cannot fully specify a system in advance -- even when using a methodology that tries to do so (Swartout and Balzar, 1982). Further, Gould and Lewis (1985) explicitly raised, and then addressed, several reasons why the principles of usability design are often not used, e.g., belief that the development process will be lengthened, belief that iteration is just fine-tuning. Human factors is more than just frosting that can be spread on at the end. "What if the development schedule is so tight that you cannot afford the luxury of talking to users," we are sometimes asked. Talking to users is not a luxury; it is a necessity. The methods described here should help with achieving a schedule. They introduce reality into the schedule, since you must do all these things eventually anyway. "Can't talking to just a few people be misleading," we are sometimes asked. Yes, possibly -- but you will be far, far better off than if you talk to none. Talking to no one is a formula for failure.

Comparison to other "User Interface Principles". The process advocated here is procedurally or methodologically oriented. In contrast, sometimes designers use the term "design principles" to refer to certain features they believe are important to incorporate into their user interfaces, e.g., icons, desk-top metaphor, consistency. Case studies that illustrate this include Xerox's Star (R) system (Smith, Irby, Kimball, Verplank, and Harslen, 1982), Visicorp's Visi On (R) (Woodmansee, 1985), some IBM Research popular PC packages (Foulger, 1986).

Case Study Evaluations. Two common threads running through reports on the development of several recent computer systems are the need for using and the effectiveness of using the design methods described here: IBM's Audio Distribution System (ADS) (Gould and Boies, 1983); Tektronix's Graphic Input Workstation (Weiner, 1984); Boeing's banking terminal (Butler, 1985); Digital Equipment Corporation's VAX Text Processing Utility (Good, 1985); Xerox's Star system (Smith, Irby, Kimball, Verplank, and Harslen, 1982); Apple's Lisa (R) system (Williams, 1983); IBM Research Computer Systems Department systems (Foulger, 1986); IBM's QMF (Boyle, Ogden, Uhlir, and Wilson, 1985). See also Akscyn and McCracken (1985). When designers follow the process advocated here, they brag about it. Not so with most other design approaches. Indeed, we have heard presentations where designers claim to have done much of what is advocated here, but clearly have not.

The development process used in OMS (Boies et al., 1985) directly tested and demonstrated the value of this process for developing a significant system. So have other case studies (Good et al., 1986; Hewett and Meadow, 1986). Because of practical limitations, these have not been controlled experiments wherein another group of designers built the same system following a different design methodology. One study, in conjunction with a one-semester course, did use a controlled experimental paradigm to compare protyping and specifying design approaches (Boehm, Gray, and Seewaldt; 1984). They found that with prototyping these student designers wrote 40% less code, required 45% less time to complete their system, and created systems that were rated higher on ease of learning and use, compared with the systems created by the students in the specifying groups. Participants in the specifying groups, on the other hand, created systems that had more functions, were more robust, had more coherent designs, and their systems were more easily integrated.

Design is a series of on-going tradeoffs among hardware, software, usability, economic, and scheduling factors. The usability design process must be followed if usability is to receive its due.

Status. If all of this is so good, why doesn't everybody do it? We previously thought that the principles of the usability design process were almost trivially simple to follow -- they are so commonsensical, and they are not difficult technically to carry out. We were wrong. They are not commonsensical to many designers (Gould and Lewis, 1985). They are hard to carry out, mainly for organizational and motivational reasons. In addition, designers in the early stages of work on a new and innovative system (a relatively rare situation in most environments, since most systems are really follow-on systems) sometimes find it hard to map the principles and methods of usability design onto their work. Once a user group is defined this becomes easier, however.

Practicing usability design is especially difficult for managers. Being willing to live in a sea of changes, which the usability design process requires, on very large projects with hundreds of people presents a significant stumbling block. Groups that report practicing the usability design process typically have a strong, committed manager. These groups are often, but not always, relatively small.

Taking the opportunity to think about the process with which you will design something rather than the design of that something, can be difficult. Designers always seem to be in the middle of something -- and never at the beginning of something with time to think about global issues.

We have met designers who are certain that the development process will be lengthened and more expensive if they practice the usability design process. They would be adding on more work, they reason, and therefore more time and effort would be required. This view fails to recognize that you learn things with this approach which eliminate a lot of work that would otherwise go on (see Boies et al., 1985; Gould and Lewis, 1985). Imagine building a house without knowing pretty much what you wanted when you started. Without a plan, you would save some time getting started, but it would certainly cost you later.

Experimental psychologists sometimes see some of our recommended methods as requiring inordinate drudge work. "I don't want to sit and watch people for hours in the experiments." Observing, listening, and making notes provide valuable insights that can be gained no other way. "Why not just completely automate computer-controlled experiments"? This automation itself requires iteration; usually there is not enough time.

To some, user testing and iterative design may seem like a pessimistic design philosophy. Do I always have to start from scratch? When will we have a scientific, analytic approach that leads to getting a good user interface right the first time? User testing and iterative design will probably always be necessary to be sure you did get it right the first time. Even expert bridge players do not always make their bids.

Nevertheless, there appears to be an increase in the use of our recommended methods, and an increased desire on the part of individual designers to try them.

Recent Advances. Two recent conceptual advances are integrated design and usability engineering. First, integrated design was added to the process of Gould and Lewis (1985) based upon the OMS experience as a test case of the other three principles (Gould, et al., 1987) (see Table 2). It may prove, organizationally, to be the hardest to follow.

Second, usability engineering, formulated at DEC (Good, et al., 1984; 1985), is a methodical, quantitative way to improve usability. Typically, the design team, in cooperation with human factors people, develops behavioral goals about how much better they want their new system to be than a competitive or previous system. For example, they may want to reduce turn-around time of a process control report by 25% or reduce hot-line questions by 20%. Then in small formal experiments, participants are given benchmark tasks to carry out on the new system. They may also carry out very similar tasks on the comparison system if these results may have not been obtained previously. These comparative results, together with other usability observations, drive the next iteration on the new system. The team tries to identify components of the new system contributing the most to the comparative results (on the targeted goals of turn-around time and hot-line calls in the example) and then improve these components. John Whiteside (personal communication, 1987), the manager of this DEC human factors group, and his group have used this approach successfully on over twenty DEC development projects. As part of this, the group is increasingly spending up to fifty per cent of their time in the field (rather in their offices or labs) visiting and observing customers.

Necessary, But Not Sufficient. Using the methods advocated here does not guarantee a GREAT system. The methods are necessary but not sufficient, as pointed out in Boies et al. (1985), to achieve acceptable usability. As in all other professions, designers have a range of ability. By definition, most systems are designed by average designers. Practicing usability design greatly increases the probability that average designers will design systems with acceptable usability. Good starting points further help (see next Section). To go beyond this and design GREAT systems requires innovation and creativity, as in the invention of the electronic spreadsheet. Also required is an outstanding leader and very good, committed people, dedication, hard work, and lots of self-imposed pressure.

New Technologies. While the usability design process is certainly useful for development projects, it is less clear how they relate to the early stages of very new research technologies and ideas. When new ideas are just beginning, they are fragile; potentially good ones can often be rejected. Imagine some new technical ideas that hardly exist today, e.g., remote control technology (maybe eventually it might be used to control or monitor your lawn mowers, children, or as a mouse replacement); locator technology (that might eventually indicate within a few feet where a particular person or object is, no matter where in the world it is; would you ever lose anything again?); a new programming technology (with which people might eventually work very differently); a brain wave recognizer (which might eventually automatically drive a typewriter or word processor, or aid in world peace negotiations); storage technology a billion or trillion times larger than today's but with "immediate" response time. The key points with these examples are that the very tough early problems are (a) to demonstrate technical feasibility apart from any application, and (b) until then it is hard to study seriously or even envision many applications that may result from a new technology (but not impossible, see Gould, Conti, and Hovanyecz, 1983).

Since the original publication of this manual five years ago, we have learned a good deal about practicing usability design on very advanced new technologies, in this case a very advanced programming technology (Gould, Boies, and Lewis, 1991).

STARTING POINTS

Where should a designer start when designing a computer system? There are several beginning points, prior to talking with potential users.

Define The System

The most important starting point is to define at least in general terms what the system will be, e.g., who will use it, what should it do, and why the users and/or organization will benefit from it. Write these down, so everyone is getting the same message. Where appropriate, define in one simple sentence the problem this system will solve for the customer. While this may seem obvious and easy, it is our observation that it is often missing, largely because of mixed motives, organizational conflicts, and focus on other things. When this is done, especially at the outset, great progress has already been made on the design problem. These goals are at least as critical in the design space as are weight, cost, MIPS, disk size, etc. Try to develop a system that users and their organizations really want.

Follow-on Systems

Most computer systems are not new. They are new releases by a vendor of existing systems (e.g., Lotus 123 (R), version 2), or extensions of already existing in-house applications. In effect, the designer is not getting started, but is already started. On the one hand the existing release of a system helps define the new design problem, but on the other hand it puts compatibility constraints on what the designer may do. The constraints may involve a trade-off between establishing positive transfer for the existing users and greatly improving the system for new users. Of importance is that the architecture of an existing system allow for easy change although it often does not.

New Influential Systems

A frequently used source of ideas for all aspects of system usability is imitation of key advances made by designers of related user interfaces, manuals, maintenance strategies, training strategies, and system functions. In user interfaces, for example, it appears that many systems have adopted the use of a mouse, icons, windows, and desktop metaphor popularized in the Xerox Star system (see Smith, Irby, Kimball, and Verplank, 1982 for descriptions of these features), or the menu style popularized in Lotus 123.

New Technologies

New technologies are often the driving force to create new systems. Examples include very large-screen displays, speech recognition, handwriting recognition, touch screens. Such projects involve innovative research. The people involved attempt to solve very difficult technical challenges. As a consequence, the work is often technology-oriented, designed to demonstrate feasibility or to contribute to the scientific disciplines from which they emanate.

While inventors want to get their ideas into use, they are not always usability oriented. It is our view that the mid-stages of the development process of new technologies can be hastened if a usability approach as described in the section on Usability Design Process is taken.

User Circumstances

Sometimes a good starting point is to build on existing user knowledge, skills, and resources. What do users expect to happen in certain circumstances? Users of computers can often describe how they wish things could be. You could talk with users in their work environments or, if appropriate, attend meetings of large user groups. In the latter case, there are official user groups associated with most large computer companies, and a list of these groups has been recently published (Datamation, 1987). Discussing advanced research technology with relevant computer users, e.g., discussing a handmarking command language with secretaries, can identify useful directions to pursue. In some large organizations marketing groups sometimes use techniques like "focus groups" to elicit some of this information. Consistent with our recommendation of direct involvement of designers with users, we recommend that designers observe these sessions, and not get the data second-hand.

Journals, Proceedings, Demonstrations

Paging through journals and proceedings can be a source of starting ideas. Journals include Human Factors, Behaviour and Information Technology, Ergonomics, Human-Computer Interaction, International Journal of Man-Machine Studies, ACM journals, particularly Communications of the ACM, and SIGCHI Bulletin. The Proceedings of the annual meeting of the Human Factors Society (e.g., 1986) and the Proceedings of Computer Human Interaction conferences (e.g., 1986) contain many relevant papers of the most recent work in the field. Such articles can be helpful, but they are usually not written in a tutorial or procedural way as are, for example, the handbooks described below. Demonstrations at national meetings, e.g., SIGCHI, and trade-shows can also be a source of ideas.

Other Designers And Consultants

Talk over your possible system with other designers. Let the attitudes of successful designers rub off on you. Benefit from lessons they have learned. These conversations can help you develop notions of good usability design and poor usability design. There are many analogies between designing good computer systems and designing other types of systems. Draw upon lessons you have learned, via successes and mistakes, in other domains. Think about how carefully you design something very important to yourself personally, e.g., a modification or addition to your house or living system. You probably have many conversations with your family (i.e., "the users") and with friends, architects, carpenters, electricians, plumbers, suppliers before going ahead with the project. Use this same sensitivity and care with the computer system you are designing.

You can obtain a list of possible consultants by looking in the Directory of the Human Factors Society. Associated with each member's name is a one-line description of their speciality and whether they are available as a consultant. You can find members living near you via the geographical area listing in it. University professors, and their students, are often interested in consulting and contract work. Try Psychology Departments and Computer Science Departments.

•
•
•

Sections on User Interface Standards, Handbooks and Guidelines, Development Procedures Books and Rules, and Formal Models for Design omitted because they duplicate material presented elsewhere in this book or because they too rapidly become out-of-date.

•
•
•

SUMMARY AND CONCLUSIONS

Usability is combination of many factors, each of which is often developed independently. User interface code is becoming an increasingly large percentage of the total system code. Standards are beginning to emerge for user interface design. Establishing standards for software aspects of user interfaces is probably premature. There are lots of guidelines for good system design. However, these are not enough for the design of good systems. You must at the very beginning and throughout development focus on prospective users and their work. We often hear that people buy computer systems for the functions in them. You are unlikely to figure out what the functions should be without talking with users. You must continuously measure each aspect of usability, and then iterate in a hill-climbing way toward a better system. All aspects of usability should begin evolving from the very beginning, and should be under one focus. Tests to date of this recommended approach lead to the conclusion that it is necessary for the design of a good system, but not sufficient. Innovation and creativity are still required to make a great system.

Acknowledgements and References ommitted in this excerpt...