

\*\*\*\*\*



**M A K I N G**  
**U S A B L E , U S E F U L ,**  
**P R O D U C T I V I T Y -**  
**E N H A N C I N G**  
**C O M P U T E R**  
**A P P L I C A T I O N S**

John D. Gould, Stephen J. Boies, and Clayton Lewis



**A**lmost a decade has passed since we started advocating a process of usability design [20–22]. This article is a status report about the value of this process and, mainly, a description of new ideas for enhancing the use of the process. We first note that, when followed, the process leads to usable, useful, likeable computer systems and applications. Nevertheless, experience and observational evidence show that (because of the way development work is organized and carried out) the process is often not followed, despite designers' enthusiasm and motivation to do so. To get around these organizational and technical obstacles, we propose a) greater reliance on existing methodologies for establishing testable usability and productivity-enhancing goals; b) a new method for identifying and focusing attention on long-term trends about the effects that computer applications have on end-user productivity; and c) a new approach, now under way, to application development, particularly the development of user interfaces.

### **Usability Design Process Review Of The Process**

The process consists of four activities [18, 20–22].

**Early Focus On Users.** Designers should have *direct* contact with intended or actual users—via interviews, observations, surveys, participatory design. The aim is to understand users' cognitive, behavioral, attitudinal, and anthropometric characteristics—and the characteristics of the jobs they will be doing.

**Integrated Design.** All aspects of usability (e.g., user interface, help system, training plan, documentation) should evolve in parallel, rather than be defined sequentially, and should be under one management.

**Early—And Continual—User Testing.** The only presently feasible approach to successful design is an empirical one, requiring observa-

tion and measurement of user behavior, careful evaluation of feedback, insightful solutions to existing problems, and strong motivation to make design changes.

**Iterative Design.** A system under development must be modified based upon the results of behavioral tests of functions, user interface, help system, documentation, training approach. This process of implementation, testing, feedback, evaluation, and change must be repeated to iteratively improve the system.

### **Status: Mixed Results**

We, and others proposing similar ideas (see below), have worked hard at spreading this process of usability design. We have used numerous channels to accomplish this: frequent talks, workshops, seminars, publications, consulting, addressing arguments used against it [22], conducting a direct case study of the process [20], and identifying methods for people not fully trained as human factors professionals to use in carrying out this process [18].

**The Process Works.** Several lines of evidence indicate that this usability design process leads to systems, applications, and products that are easy to learn, contain the right functions, are well liked, and safe. The process is now well known; nearly all human factors people endorse it. Several others, often working independently, have advocated more-or-less similar ideas: Bennett [3]; Bury [9]; Damodaran, Simpson, and Wilson [13]; Meister [28]; Reitman-Olson [31]; Rubinstein and Hersh [33]; Shackel [34]; the Usability Engineering group at DEC including Whiteside, Good, and Wixon (e.g., [17, 40]).

Second, the usability engineering process developed at DEC by Whiteside and colleagues has been reported to lead to many successful applications [17].

Third, retrospective reports about the development of computer systems, including several that have been very influential, emphasize the value of following

this type of development process: Xerox's Star system [35]; Apple's Lisa™ system [39]; IBM's Audio Distribution System (ADS) [19]; IBM's Rexx [12]; Tektronix's Graphic Input Workstation [36]; Boeing's banking terminal [10]; Digital Equipment Corporation's VAX Text Processing Utility [16]; IBM's QMF [8]; IBM Research Computer Systems Department systems [15]; Lotus Development Corporation's Lotus 1-2-3 [26]. Also see [1].

Fourth, case studies have attempted to test, and have successfully demonstrated, the value of this process [9, 17, 19, 25]. Because of practical limitations, these have not been controlled experiments wherein another group of designers built the same system using a different development process. One study, in conjunction with a one-semester university course, did use a controlled experimental paradigm to compare "prototyping" and "specifying design" approaches [7].

**But, The Process is Not Often Used.** Certainly, most designers want to create useful, usable, likeable systems. So, why is this process, which has existed for a decade or so, not used more often? Experience has shown that there are powerful, interacting organizational and technical reasons.

First, development work is organized around goals that are to be met. These generally do not include *usability* goals.

Second, despite the evidence [e.g., 17], there is still a belief that usability cannot be measured. Aside from programmer productivity studies, there are no generally accepted behavioral metrics, even rough ones, that describe the trends in the usability of computer systems in the last two decades (but see [14]).

Third, also playing a role are management reward structures and apparent conflicts between meeting deadlines and achieving usability. Project management is often reluctant to sign up for usability because



of lack of confidence in managing something that does not have clear goals and that does not have tools to efficiently address problems as they arise. Indeed, management has to solicit others—all of whom already have more than enough to do—to carry out most recommendations that usability people make.

Fourth, designers report that software development is not organized to carry out this process. Iterative design is thought to be too risky, too difficult, and too time-consuming. The user-interface code and the functional code are intermixed. Because of the amount of effort required and unanticipated side effects, application managers often want to minimize user-interface changes—even though these changes would probably enhance usability.

Fifth, designers need better tools in order to do iterative design.

Sixth, nearly every new application creates its own user interface. The work is enormous (often over half the code), is not leveraged (usually does not benefit from code reuse), and is generally not carried out by people skilled in user-interface design.

#### **What To Do?**

What is needed for this process of usability design to be used more often? How can this process, which works so well for small groups of outstanding people having dynamic leaders, be made to work in much larger work organizations and in organizations which have a higher proportion of workers with average abilities? The original contribution of this article is to address these questions by outlining possible solutions to the six problems just mentioned.

#### **Testable Usability Goals For Individual Systems**

Many components of systems and applications have technical goals which help focus the development work (e.g., cycle time will be 20 nsec; the system will have no more than 2 hours of downtime per

month; the memory must store 4 megabytes of data). Existing methodology shows that analogous goals can be established for usability, as illustrated in the next two examples.

#### **Example: Testable Usability Goals In Existing Systems**

Table I is adapted from the monthly news bulletin of the IBM Research Computer Center. The rows show four usability measures. Three of these are direct measures of system performance which also are key indicants of user satisfaction and performance. The fourth is a subjective rating obtained daily from users when they logoff. The last column contains the usability goal (target value) for each measure. These have been established by user committees and the computer center staff, and have evolved over the years. The other columns contain one month's data on the performance of three large time-shared computer systems used by hundreds of users each day.

These goals are clearly stated, easily communicated, and verifiable. Continuous system monitoring is conducted through automatic data logging. The results are in *public* use, published monthly. Attention centers on them. Experience shows that the results are taken seriously by users and by management. For instance, hardware is ordered when goals are not met.

#### **Example: Testable Usability Goals In New Systems**

Analogous measurable goals have been established for end-user performance on applications under development, particularly with the Usability Engineering approach developed at the Digital Equipment Corporation [17, 40]. Table II provides an example. The design team makes a list of the desirable usability attributes (column 1). One of those is installability. The team then decides how installability will be measured. In the example in Table II, they decide it will be measured with a laboratory task. Time to in-

stall will be the dependent variable. But what should the goal, or target time be? The designers discuss it. Since nearly all computer applications are follow-ons of existing applications, the design team has an opportunity to survey today's situation ("Many cannot install it by themselves"). They decide on the worst case they would be willing to put their name on (1-day with media), and the best case given sufficient time and resources ("10 minutes with media"). They then set their goal ("Planned Level" of Table II) to be one hour. This process is repeated with each of the other usability attributes. These discussions focus the energy of the design team in the right places.

The key points to note are that this is a group process of setting goals, deciding what the relevant usability attributes are, how to measure them, and what the target goals should be. The goals are clearly stated, easily communicated, and verifiable throughout the development process. Measurement is implied. Project management has these goals available, just as they have goals for other system components, and can therefore manage usability in similar ways. The process provides numeric results, which management likes. Results with respect to meeting these measurable goals can be incorporated into product forecasting.

For the purposes of this article, the important implications of these two examples are that usability goals can be organizing forces for development work, and of necessity can stimulate iterative design if they are to be met.

#### **Focusing Attention On Long-Term Trends In End-User Productivity**

We have just discussed impacting the usability of *individual* systems. Now let us shift to planning for usability or productivity-enhancing applications on a broad scale, independent of individual systems. Perhaps surprising to some, with the exception of programmer produc-



tivity studies, there are almost no quantitative behavioral measures of general trends, over the years, on how human and organizational productivity are directly affected by the use of computer systems (but see [14]). As a result, research and development work is not organized in ways that it would be if such knowledge existed and was taken seriously. There is an uncertainty and lack of salience about human and organizational productivity trends when making, purchasing, or using computers. Computer vendors and customers cannot take into account, in a long-term way, trends in human and organizational productivity because they are unknown. This lack of general metrics, and their use, suggests that the usability and productivity of people and organizations who use computers is not a serious goal for much application development.

**Analogies Outside The Computer Industry**

There are behavioral measures of general trends and progress in other "industries." For example, in automobile and highway design, there are measurements of accidents per thousands of miles driven. In disease control, there are measurements of the number of incidents per thousands of citizens. In education, there are achievement test scores. These measures are tabulated annually. Each provides a record of past history, an indication of the present situation, and can be used to project the future. Goals for the future can be set in terms of these measurements, and whether or not these goals are met can be determined. Even if the measures are controversial, as are SAT scores in education, they get the focus on "how are we doing" with respect to "users."

**Analogies Within The Computer Industry**

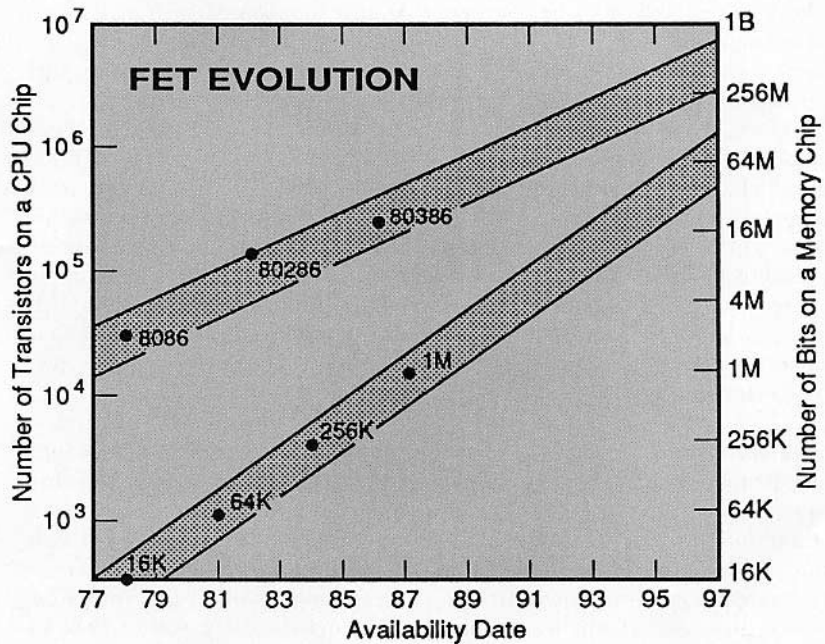
Within the computer industry, there are measures of general progress for some major system components. These provide a basis

**TABLE I. Usability Goals for Existing Systems.**

System Performance Data—Adapted from 2/88					
PERFORMANCE	SYS1	SYS2	...	SYS5	GOAL
Percent Available	100.0	99.9		99.2	97.0
Downs/Day	0.00	0.10		0.10	<=0.50
90 Percentile Response Time (sec.)	0.25	0.15		0.16	<=0.35
Average User Satisfaction Rating	2.6	2.15		2.32	<=2.5
... from IBM Research Center Computer Center					

**TABLE II. Usability Goals for New Systems.**

Attribute	Measuring Concept	Measuring Method	Worst Case	Planned Level	Best Case	Now Level
Installability	Install Task	Time to Install	1 Day with Media	1 Hour without Media	10 Minutes with Media	Many cannot Install
Learning Rate						
Fear of Seeming Foolish						
... From Digital Equipment Corporation						



**FIGURE 1. Increases in Chip Density and Miniaturization. Just as Hardware Components are Tracked, Usability Components of Computer Systems may also be Tracked and Projected: FET = Field Effect Transistor; CPU = Central Processing Unit.**



for long-term planning of these components. Figure 1 provides an example, showing how the number of transistors on an integrated circuit chip made for the central processing unit (CPU) of computers has increased over the years. (Similar graphs show long-term trends of other important computer systems components, for example, communication costs, communications bandwidth, system response time, disk access time, system reliability.)

Such graphs have easily measured, sensible dependent variables. They provide such information about a technical field as where it was, where it is now, and where it will be—either through one company's efforts or another's. Of real significance, they are not proprietary, but are shared among scientists and engineers throughout an industry—all of whom are competing to do better. They are based upon the efforts of many companies and universities. They show huge improvements over decades. These improvements do not just happen, but are in part driven by the knowledge that to be competitive one must find ways to make new products reach the projected levels of improvement, as in Figure 1. These graphs are taken very seriously by management. They contain believable metrics. They stimulate decisions about how to organize new work. They drive investment, which is needed to stay on the projected curve. They assume that results will be measured. They demand innovation to stay on the projected curve.

#### **The New Idea**

We propose that analogous analyses and graphs be developed to understand long-term *trends* in the *usability* of computer systems and in human and organizational productivity that result from using these systems. Such analyses can set goals for future research and development that will affect usability. Just as they have done in other technical areas, these analyses would proba-

bly stimulate both a) investment—in this case to impact human and organizational productivity using computers—and b) the working relations among people trying to produce these systems. They would also probably raise the level of accountability of both management and usability people to meet human and organizational productivity goals. There is a view that computers have not increased the productivity of individual users and their organizations despite their widespread use and cost [2, 27]. Such trend analyses are needed to ascertain whether or not progress is being made, and if so, what the rate of improvement is. The result: the consumer benefits.

Proprietary analyses now exist showing improving *trends* in code reliability, hardware reliability, and vendor installation times—in effect, trends in *vendor employee* productivity. These improvements happened because of goal setting, measurement, feedback, management attention, improved tools and methods, and work reorganization. What is being called for here, then, is an extension of such analyses to *customer* or *user* productivity.

What is needed to make such trends-in-user-productivity graphs, analogous to Figure 1? First, thoughtful benchmark tasks. These would reflect individual and organizational tasks that have a long history and a likelihood of continuation, for example, “document” composition; communication within an organizational hierarchy, financial analyses, transaction processing.

Second, appropriate dependent variables are needed. One candidate set reflects human performance on common user tasks that computers could, or do, help with. A second candidate set involves measuring group and organizational productivity, rather than individual productivity. A third set of candidates are system variables that can be shown to bear at least a monotonic relation to human productivity and feelings. Possibilities in-

clude system response time, availability, and human-computer transaction rate [14]. A fourth candidate set involves measuring general cognitive variables (e.g., mental effort or stress) and affective states while people do computer-related work.

Third, estimates of the past, measurements of the present, and projections of the future with respect to these benchmark tasks and dependent variables are necessary.

Established methodology does exist for managing and measuring the usability of individual users while using individual computer systems, as illustrated in the above section on testable usability goals. This needs to be extended in three ways: from measurement of ease of learning to measurement of productivity; from measurement of individuals to measurement of organizations; and from measures of individual systems to general metrics cutting across many systems and many years.

#### **Example**

To illustrate how this might work, consider a hypothetical example, regardless of how imperfect. Document composition has been and remains a common user task and therefore a reasonable basis for benchmark tasks. Document composition *rate* is a reasonable dependent variable. Assume that document composition rate has through computer aiding increased at 1 word/min/year over the last 15 years, and is now up to, say, 30 words/min. Assume that this trend is predicted to continue for the next decade, and perhaps beyond. What kind of technology must we invent to be competitive? What aids will allow people 10 years from now to compose at 40 words/min? These questions must be taken seriously because the prediction—from experience with other system components modeled as in Figure 1—is that someone will come up with a solution. Should the emphasis be on making systems with faster human output methods, for exam-



ple, speech; on revision aids; on appending or amending rather than composing from scratch; etc.? These questions focus the discussion of possible new technologies on exactly the right issue: how each might affect usability, human productivity, and organizational productivity.

Don't be too harsh on this example. Try a similar one for tasks now carried out with spreadsheets, using errors and maybe time as the dependent variables; or for transaction tasks, using time as a dependent variable.

### **A New Approach To Application Development**

We earlier identified six reasons why usability design is not much used. The first two and part of the third have just been addressed. The remainder are addressed here:

- Iterative design is too risky.
- There is a need for better software tools.
- Each application generally redoes the user interface.

These problems are fundamental, difficult to solve, inhibit the development of innovative user interfaces, and limit the growth of the computer industry. Providing significantly better software tools is not just a matter of inserting them into present development processes; instead a new approach is needed.

At the IBM Research Center an interdisciplinary group of computer science and human factors people is trying to address these problems in a project called ITS [7]. As described below, we have invented, implemented, and demonstrated the feasibility of a radically different technical approach, which leads to a serious reorganization of software development work. Initially we are using this approach ourselves as well as working with potential users ("customers") of this approach, letting their requirements and feedback drive our research energies.

### **Lessons From The History Of Work**

*In General.* By way of background, throughout history the evolution of every industry demonstrates the interdependence of the *organization of work*, *available tools*, and *work roles*. Historically, tasks have been subdivided and component parts modularized to increase productive efficiency. This has led to a division of labor and increasing work specialization. The speed and cost of designing, engineering, and manufacturing complex systems today, as always, depends upon how the work is organized, the ability to utilize relevant previous work (e.g., how little retooling or reprogram-

all aspects of it (Figure 2a). Today, they no longer need to program their own operating system, file manager, database manager, query language. Instead, these "components," and many other library-selectable routines, have been previously programmed by others and are available for general use. Software tools seriously modify work organization [4].

*User Interfaces.* User-interface design has not benefitted much from modularization and code reuse. The result is that today in many applications half or more of the new code is user-interface code. If this large programming mass could be decomposed into compo-

**The evolution of every industry demonstrates the interdependence of the organization of work, available tools, and work roles.**

ming is necessary), and how much new work can go on in parallel (e.g., the degree to which the new machine, system, or user interface is modular). Standardized parts and interfaces, and the evolution of larger premade modules, are beneficial.

*Computer Systems.* In its brief history and extraordinarily rapid evolution, the development of computer applications has undergone a huge degree of work specialization and modularization. To create a new application in the early years, application developers "started from scratch"; they programmed

nents which allowed specialists in each component (including non-programmer specialists) equipped with appropriate tools, to work in parallel and use, rather than duplicate, the completed contributions of each other, a huge gain in productivity and usability could result.

### **ITS (A Rapid Application Development System)**

In our attempt to provide a framework in which application developers can more readily follow usability design, we have been influenced by these observations about the evolution of human work. Here we dis-



cuss several key concepts of ITS: four end-user operations; separation of the content and the style of applications; rule-based and computer-executable styles; four independent organizational and architectural components; and four work roles.

**Four End-User Operations**

Our informal analyses of a variety

of user interfaces indicates that end-user activity involves four operations: filling in forms; selecting among prescribed choices, manipulating lists, and reading information. This insight leads to four corresponding building blocks that are sufficient to abstractly describe user interfaces:

- Form blocks,
- Choice blocks,

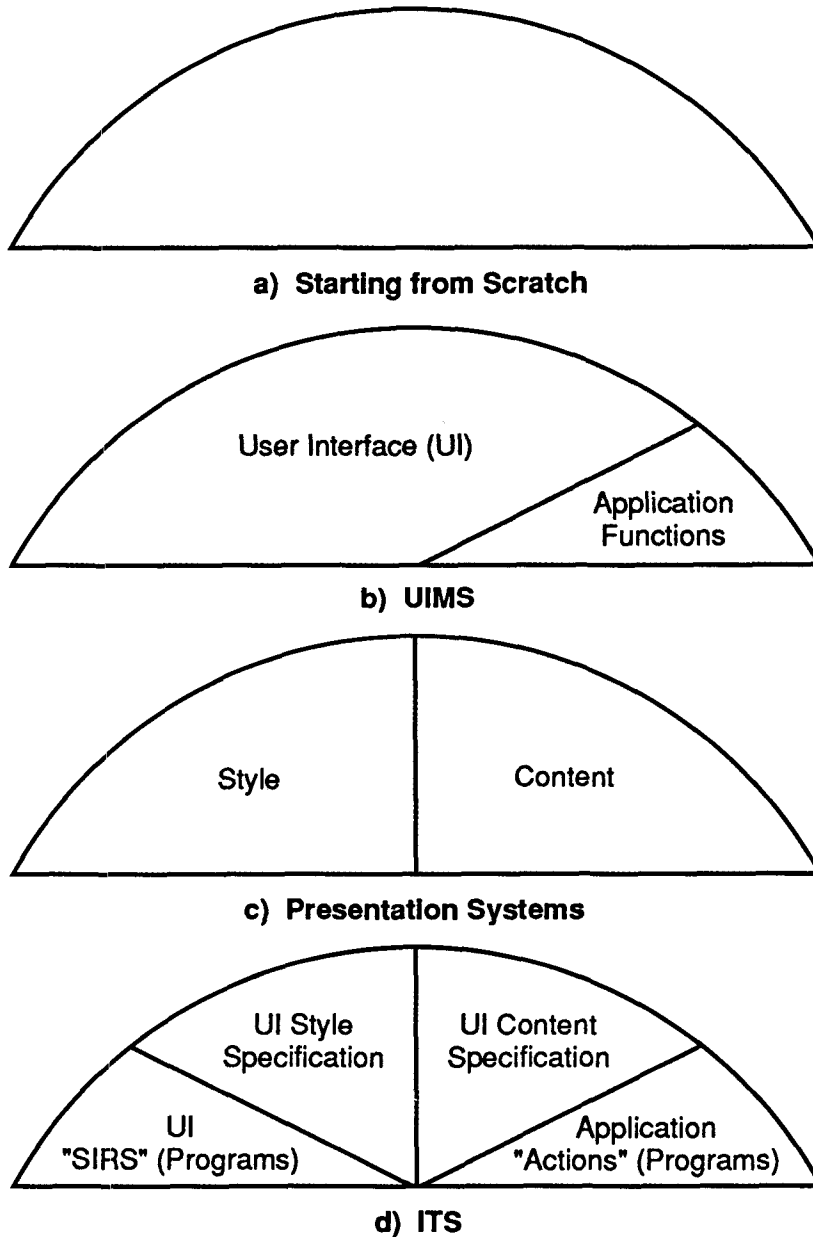
- List blocks, and
- Info blocks.

With ITS, application experts (see below) structure their applications in terms of these form, choice, list, and info blocks. Style designers (see below) write rules about how these blocks will be rendered on an end-user's screen under various circumstances. These two groups can work in parallel and independently.

**Separation Of Style And Content**

**UIMS.** Work on "user interface management systems" (UIMs) has demonstrated that it is possible to separate the user interface from functional code (e.g., [24, 29, 30] (Figure 2b). This separation is valuable because it reduces the risks when designers make changes to the user interface, therefore promoting iterative design. It also can promote code reuse. For example, several different user interfaces could make use of the same functional code. On the other hand, this approach does not lead to consistent user interfaces across many applications. Most UIMS work is presently at the research stage; this separation appears in relatively few real-world applications (e.g., [16]). Further, there is some skepticism that UIMS, as presently implemented, can produce today's advanced user interfaces involving direct manipulation with rich feedback [32].

**Presentation Systems.** Fundamental to achieving consistency across applications is the need to separate what might be called the "style" of an application (e.g., the look, feel, and workings of human-computer interaction techniques shown on the user interface) from the "content" of an application (e.g., the substance—the information that differentiates a passenger reservation application from an insurance agent's application). Apple Corporation's Macintosh and IBM's CUA are examples of application style. "Presentation Systems" (e.g., Presentation Manager (PM), Mac-ToolKit) make this content-style



**FIGURE 2.** Different Approaches to Application Development: a) Starting from Scratch; b) UIMS; c) Presentation Systems; d) ITS.



separation (Figure 2c). Typically, these systems supply several human-computer interaction techniques (called “controls” in PM), each in one, fixed style (e.g., format, color, highlighting, behavior). This approach leads to a consistent style of user interface across several applications. However, to make even a small change in a style requires a serious programmer. Further, an application can work only in the style for which it is written. An application written for the Macintosh style, for example, will not work in the IBM CUA style (which uses PM). Thus, while there is the benefit of code reuse through the reuse of style, there is no benefit of reuse of the application content. Further, a knowledgeable programmer is needed to “connect” these controls to an application’s content.

**ITS.** With ITS (Figure 2d), we are developing an approach that incorporates both the separation that UIMs aim at and the separation that “Presentation Systems” aim at. ITS decomposes the user-interface specifications and code into content and style, keeping both separated from the functional code. ITS is an environment in which many different styles work.

On the one hand, with ITS, the content of many different applications can be run in the same style. There is no need to redo the style code every time a different application is developed. This approach should lead to consistent user interfaces and facilitate users learning later releases of existing applications, or new applications. Besides greatly reducing the amount of new work through style reuse, this approach has the advantage of leveraging the work of the best style designers.

On the other hand, in ITS the same content application can run in several different styles. This has the advantage of greatly reducing the amount of new work (e.g., the same excellent registration application, for example, could run in one style at one university, a different style at

a second university, and a different style at the census bureau). It also leverages the work of the best application designers (those who create applications with insightful functions that lead to enhanced productivity for end users and their organizations) across a variety of end-user organizations.

Today many computer vendors (e.g., IBM, Apple, DEC) as well as many of their customers (e.g., airlines, insurance companies, hotels) have their own individual user-interface styles (sometimes called “user-interface standards”); the purpose is to a) ensure consistency within and across their applications; b) reflect the taste of key people; and c) have their own “image.” As a result, each application must now be programmed in a company’s own style. Because style and content are not separated, a vendor, for example, who wants to make an application for several companies may have to redo the entire application for each company. Having done that, if the vendor now wants to make a second application for these same companies, he or she must repeat this process, unable to use much, if any, of the style programming for each company from the first application. ITS solves these problems.

Thirdly, with ITS the same application—content and style—can run on several different operating systems (ITS has so far run on DOS, OS/2, and AIX). This has the advantages of economy of scale, reduced need to do an application over for each operating system, and making the best work available to a larger number of individuals and organizations.

#### **Rule-Based, Computer-Executable Styles**

Today developers have great difficulty locating, reading, understanding, recalling, and applying the massive amounts of required detail contained in user-interface guidelines books. With ITS, guidelines books can be eliminated. With ITS, style rules are executable and

are instantiated in computers rather than in books. A new application is *automatically* rendered in one or more styles. This allows very rapid prototyping—typically some progress is made within an hour. Styles are not tuned to a specific application, but contain general, executable rules controlled by a set of parameters. Each style has a different set of rules which a) map content into human-computer interaction techniques and b) determine the details of each interaction technique. To illustrate (Figure 3), a style could render the same set of choices (e.g., which let users select Freshman, Sophomore, Junior, or Senior) as an action bar, as a menu, or as a half-moon. Within the same application, this choice block might be rendered one way in one context and another way in another context.

#### **Four Organizational And Architectural Components**

ITS has four organizational and architectural components. Two of these are on the content side and two are on the style side (Figure 2d):

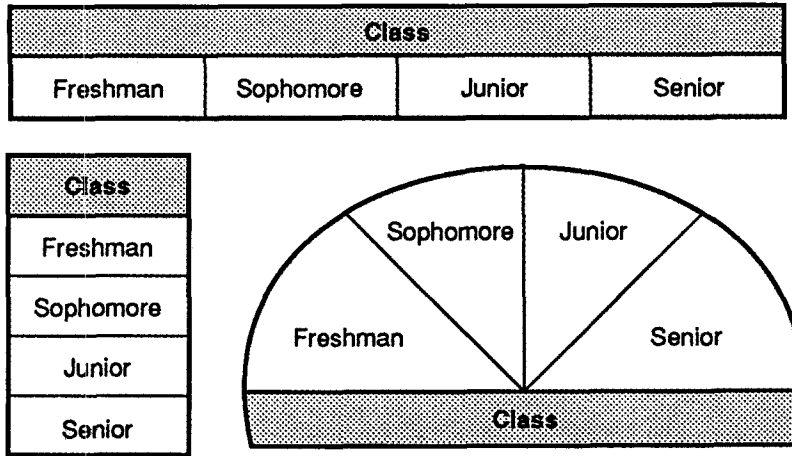
*Content (application) specifications*, which include the messages to end users, flow of control, connections to function, and guidance to style. These are created by application experts (see below).

*Content (application) actions*, which are atomic programs with general utility, for example, transfer the contents of one list to another; add a string of numbers. These are created by application programmers (see below).

*Style specifications*, which are the rules regulating the set of human-computer interaction techniques used to render content, including interaction methods (e.g., entry vs. selection), appearance of the end-user’s screen, and the interaction devices. These are created by style designers (see below).

*Style programs* make a particular interaction technique work. These are created by style programmers (see below).





**FIGURE 3.** Three Renditions of the Same Set of Choices: Menu Bar (top); Standard Menu (left); and Half-Moon (right).

#### Four Work Roles

Each work role is associated with a different one of these four components. The linkages among these four work roles are very different than those which exist today. Experience to date indicates they provide the potential for rapidly producing usable applications.

*Content* or *application* experts exist for nearly all new applications. (We use the two terms content expert and application expert interchangeably.) For example, registrars know their schools' rules and policies for student registration. They understand the "content," (i.e., the work or business that is basic to their existing or new application). They know the jobs of possible end users. They can identify the data that must flow across the user interface—to and from end users. Today, however, their role is limited to being interviewed, indeed often by intermediaries only, rather than by the developers themselves. Given appropriate tools, application experts (assisted by "analysts" if needed) can structure their knowledge and instantiate it in computer-executable form. They do not need to know much about user-interface style. Application experts can be assisted by human factors people to make usability requirements more salient

and to do user testing.

Application experts create the user-interface content specifications (Figure 2d). Figure 4 shows an example display of an action bar across the top, a pull-down menu from the choice item called "File," and a window containing a scrollable list and three pushbuttons. The cloud on the middle right of Figure 4 illustrates schematically what an application expert did to create the window. Having application experts involved directly increases the probability that the resulting application will contain the right functions to enhance end-user, and their organization's productivity.

*Application (content) programmers* get from others, or write themselves, any new actions (programs) required by their application (Figure 2d). To date, they have programmed in C. The cloud on the bottom right of Figure 4 schematically illustrates this.

*Style designers* also exist today, having skills in human factors and graphic design. Their current role is mainly that of advocate; they identify problems and describe solutions, but do not have the tools to implement their ideas during application development. (They do have prototyping tools, but the results of that work are generally not directly

incorporated into their applications.) With ITS, *style designers* specify style rules (Figure 2d). They select a human-computer interaction technique from the existing ITS library, and they compose general rules about the content conditions under which that interaction technique will be used. The cloud on the middle left of Figure 4 illustrates the spirit of some style rules to render a choice block as push-buttons. Style designers create styles that are suitable for a variety of applications.

*Style programmers* are needed to create entirely new interaction techniques, if a style designer cannot modify an existing one to get the desired result. Style programmers write programs necessary for making an interaction technique work (the "SIRS," or style implementation routines, in Figure 2d). To date, they have done this in C. The cloud on the lower left of Figure 4 schematically illustrates this.

#### Tools

ITS provides content experts with tools to help them structure their knowledge about their application in terms of the four building blocks (choices, forms, lists, and information blocks). Style designers have tools to select appropriate human-computer interaction techniques from a computer-based library of well-tested ones, and to map these onto an application's content blocks. Presently these are markup languages, as suggested in Figure 4. A WYSIWYG approach is not used because of the need to specify this information in an abstract way [38]. We are using ITS to create "workbenches" for these experts, i.e., using ITS to create user interfaces for ITS tools.

#### Status of ITS

This radically new way of thinking about computer applications development is not just a pipe dream. In the last three years at the IBM Research Center we have been designing and implementing these ideas, and demonstrating their feasibility



[7, 37]. We have given over 100 live demonstrations of ITS to a variety of people and in a variety of meetings. So far, major parts of several styles have been created, including IBM's CUA-2 style. User interfaces, based upon the (content) requirements of 20–30 applications, have been created, many in a few days each, and some have been demonstrated in multiple styles. Present emphasis of the work includes a) developing further core ITS; b) demonstrating ITS to both computer scientists involved in research and to real-world application developers, getting their feedback and understanding their requirements; c) helping various application groups use ITS to develop their own applications, and being driven by their feedback; d) carrying out applications of our own. ITS is a true rapid application development system. A prototype made with ITS becomes part of the final application itself. This is in contrast to the use of most popular prototyping tools available today, which require developers to begin again, once the prototype is complete.

The major aims of ITS are to provide designers and developers with an organizational and technical approach with which they can be more productive and can create applications which allow their user organizations to be more productive. Such applications require iterative design, and our experiences to date demonstrate the ability of non-programmers to iteratively improve the user interface.

Experience shows that the three-way split among style of the user interface, content of the user interface, and the functional code allows changes to be made in the user interface and still preserves the integrity of the functional code. Iterative design—so necessary to achieve good usability—proceeds rapidly, without the usual risks because of the way the work is decomposed.

On the style side, a particular style can be prototyped and iteratively engineered. In the long run, winning styles can emerge from a

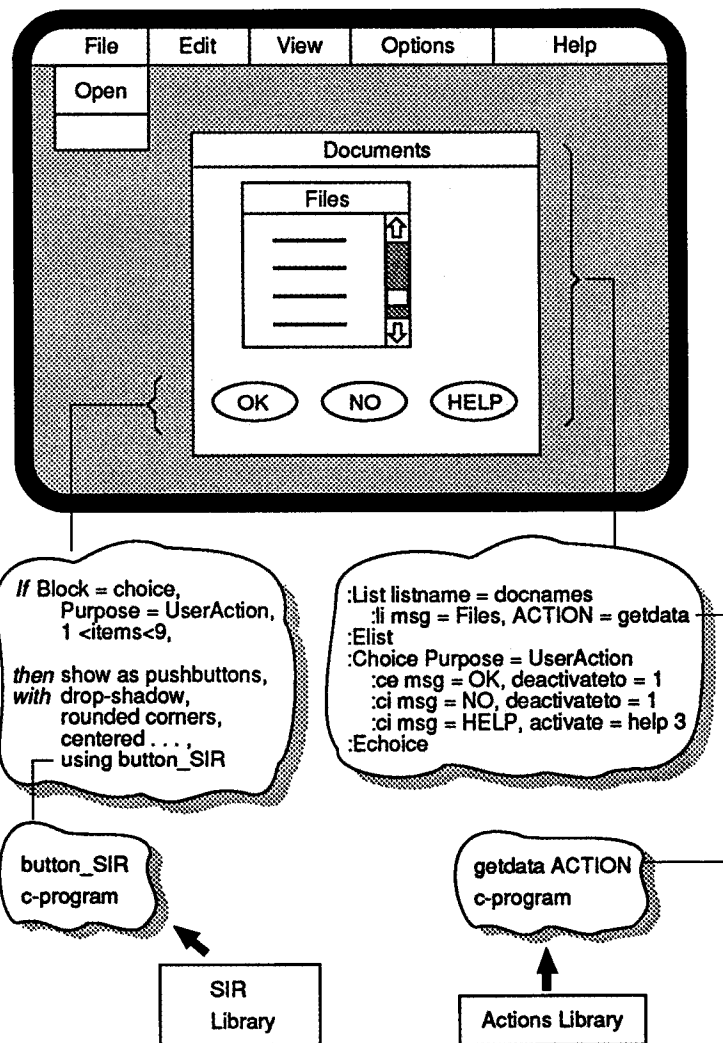
larger set of styles, the dropouts having never attained the favor of organizations or end users. This approach will leverage the work of the best style designers.

ITS allows parallel development of application components (just as automobile design allows parallel development of tires, ignition system, and braking system). ITS allows exciting new human-computer interaction techniques to be incorporated into an existing style file. ITS provides a mechanism whereby

the results and details of laboratory human factors experiments can be incorporated into a widely used style library [23]. It appears that ITS will make it much easier to maintain existing applications (once written in ITS) over the years, a job that is very difficult and expensive today [11].

### Conclusion

The process of usability design is vital to achieving usable, likeable, productivity-enhancing applica-



**FIGURE 4.** Schematic illustration of how each of the ITS Work Roles contribute to an Application. Application Experts Specify the Content of the User Interface (upper-right cloud); Content Programmers Write the Functions or Actions (bottom-right cloud); Style Designers Write the Style rules (upper-left cloud); and Style Programmers Write the Style Implementation Routines (bottom-left cloud).



tions. Nevertheless, the process is not used as much as many designers would like, due to organizational and technical reasons. The goal of this article was to identify the problems limiting the use of usability design in creating computer applications, and then suggest solutions to them. Three solutions were described: a) greater reliance on existing methodologies for establishing testable usability and productivity-enhancing goals for individual applications; b) a new method for identifying and focusing attention on long-term trends about the effects that computer applications are having on end-user productivity; c) a new technical and organizational approach to application development, now under way, that directly involves application experts and style designers and makes iterative design much easier than heretofore. **□**

#### Acknowledgments.

This article is based upon the 1988 Human Factors Society Presidential Address of John Gould. Gould worked jointly with Clayton Lewis on the development of the usability design approach, and with Stephen Boies, who is the director of the ITS project, on most everything.

#### References

1. Akscyn, R.M. and McCracken, D.L. Zog and the USS CARL VINSON: Lessons in system development. In *Proceedings of Interact'84 First IFIP Conference on Human-Computer*. Elsevier Science Publishers, Amsterdam, 1985, 303-308.
2. Attewell, P. The productivity paradox. 1990. Unpublished manuscript.
3. Bennett, J.L. Managing to meet usability requirements: Establishing and meeting software development goals. In *Usability Issues and Health Concerns*, J. Bennett, D. Case, J. Sandelin, and M. Smith Eds. Prentice-Hall, Englewood Cliffs, N.J. 1984, 161-184.
4. Blomberg, J.L. The variable impact of computer technologies on the organization of work activities. In *Proceedings of the Conference on Computer-Supported Cooperative Work* (Austin, Tex., 1986), pp. 35-42.
5. Boehm, B.W. A spiral model of software development and enhancement. *IEEE Comput.* (May 1988), 61-72.
6. Boehm, B.W., Gray, T.E., Seewaldt, T.. Prototyping versus specifying: A multiproject experiment. *IEEE Trans. Softw. Eng.*, 10, 3 (1984), 290-302.
7. Boies, S.J., Bennett, W., Gould, J.D., Greene, S.L., and Wiecha, C. The interactive transaction system (ITS): Tools for application development. IBM Res. Rep. RC 14694, 1989.
8. Boyle, J., Ogden, W., Uhler, S., Wilson, P. QMF usability: How it really happened. In *Human Computer Interaction—Interact '84* (1984), pp. 877-882.
9. Bury, K.F. The iterative development of usable computer interfaces. In *Human-Computer Interaction—Interact '84* (1985), pp. 343-348.
10. Butler, K.A. Connecting theory and practice: A case study of achieving usability goals. In *Human Factors in Computing Systems Proceedings CHI'85* (1985), pp. 93-98. ACM, N.Y.
11. Corbi, T. Program understanding: Challenge for the 1990s. *IBM Syst. J.*, 28, 2 (1990), 294-306.
12. Cowlshaw, M.F. The design of the REXX language. *IBM Syst. J.* 23, 4 (1984), 326-335.
13. Damodaran, L., Simpson, A., Wilson, P. *Designing Systems for People*. Manchester, England, NCC Publications, 1980.
14. Doherty, W.J., Thadhani, A.J. The economic value of rapid response time. IBM Tech. Rep. GE20-0752-0, 1982. (Available from author at IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598.)
15. Foulger, D. Discovering the PC User Interface. Unpublished Manuscript, 1986. (Available from the author at IBM T.J. Watson Research Center, Yorktown Heights, N.Y. 10598.)
16. Good, M. The iterative design of a new text editor. In *Proceedings of the Human Factors Society—29th Annual Meeting* (Santa Monica, Calif, 1985), pp. 571-574.
17. Good, M., Spine, T.M., Whiteside, J., George, P. User-derived impact analysis as a tool for usability engineering. In *Human Factors in Computing Systems CHI'86 Proceedings* (1986), ACM. New York, pp. 241-246.
18. Gould, J.D. How to design usable systems. In *Handbook of Human-Computer Interaction* M. Helander Ed., Elsevier Science North-Holland Publishers, 1988, 757-789.
19. Gould, J.D. & Boies, S.J. Human factors challenges in creating a principal support system—The speech filing system approach. *ACM Tran. Office Info. Sys.* 1, 4 (1983), 273-298.
20. Gould, J.D., Boies, S.J., Levy, S., Richards, J.T., Schoonard, J. The 1984 Olympic Message System—A test of behavioral principles of system design. *Commun. ACM* 30, 9 (Sept. 1987), 758-769.
21. Gould, J.D., Lewis, C.H. Designing for usability—key principles and what designers think. In *Proceedings of the 1983 Computer-Human Interaction Conference* (1983), pp. 50-53.
22. Gould, J.D., Lewis, C.H. Designing for usability—key principles and what designers think. *Commun. ACM* 28 (1985), 300-311.
23. Greene, S.L., Gould, J.D., Boies, S.J., Meluson, M. and Rasamny, M. Evaluation of entry and selection methods in an airlines reservation scenario. *IBM Res. Rep., RC-15799*. 1990.
24. Hartson, R. User interface management control and communication. *IEEE Softw.* (Jan. 1989), 62-70.
25. Hewett, T.T., Meadow, C.T. On designing for usability: An application of four key principles. In *Human Factors in Computing Systems CHI'86 Proceedings*, (1986), ACM, New York, pp. 247-252.
26. Kapor, M. Invited talk, IBM Research, Hawthorne, New York, 1989.
27. Landauer, T.K. Human factors driven design of information retrieval systems. Invited address, Human Factors Society, Denver, 1989.
28. Meister, D. *Human Factors Testing and Evaluation*. North-Holland Science Publishers, Amsterdam, Netherlands, 1986.
29. Olsen, D.R., Kasik, D., Rhyne, J., and Thomas, J. ACM SIGGRAPH workshop on software tools for user interface management. *Comput. Graph.* 21, 2 (1987) (see papers following this introduction), 71.
30. Pfaff, G.E. Ed. User interface management systems. In *Proceedings of the IFIP/EG Workshop on User Inter-*



face Management Systems. Seehiem, (Federal Republic of Germany, October, 1983) Springer-Verlag, N.Y. 1985.

31. Reitman-Olson, J. Expanded design procedures for learnable, usable interfaces. In *Proceedings of the ACM-SIGCHI Human Factors in Computing Meeting*. (1985), ACM, New York, pp. 1-3.
32. Rosenberg, J., Hill, R., Miller, J., Schulert, A., and Shewmake, D. UIMSs: Threat or menace? In *Proceedings of CHI*. (Washington, May 15-19, 1988), ACM, New York, pp. 197-200.
33. Rubinstein, R., Hersh, H. *The Human Factor: Designing Computer Systems for People*. Digital Press, Burlington, Mass., 1984.
34. Shackel, B. Information technology—A challenge to ergonomics and design. *Behav. Inf. Tech.* 3 (1984), 263-275.
35. Smith, D.C., Irby, C., Kimball, R., Verplank, B., Harslem, E. Designing the star user interface. *Byte* 7, 4 (1982), 242-282.
36. Weiner, H. Human factors lessons from the design of a real product. TEK Tech. Rep., 1984. (Can be obtained from the author at Tektronix, Box 1000, Wilsonville, Oregon 97070).
37. Wiecha, C., Bennett, W., Boies, S., and Gould, J. Generating highly interactive user interfaces. In *Proceedings of CHI'88* (Austin, Tex., April 30-May 4, 1989). ACM New York, pp. 277-282.
38. Wiecha, C., Boies, S., Green, M., Hudson, S., and Myers, B. Panel: Direct manipulation or programming: How should we design interfaces? In *Proceedings of the Second Annual ACM SIGGRAPH Symposium on User Interface Software and Technology*. ACM Press, N.Y., pp. 124-126.
39. Williams, G. The Lisa computer system. *Byte* 8, 2 (1983), 33-50.
40. Wixon, D. and Whiteside, J. Engineering for usability: Lessons from the user-derived interface. In *Human Factors in Computing Systems CHI'85* (1985), pp. 144-147. ACM, N.Y.

**CR Categories and Subject Descriptors:** C.5 [Computer Systems Organization]: Computer System Implementation; D.2.2 [Software Engineering]: Tools and Techniques—User interfaces;

H.1.2 [Models and Principles]: User / Machine Systems—Human factors; K.6.1 [Management of Computing and Information Systems]: Project and People Management—Systems analysis and design, systems development; K.6.3 [Management of Computing and Information Systems]: Software Management—Software development

**General Terms:** Design

**Additional Key Words and Phrases:** Application development, usability, usability engineering

**About the Authors:**

**JOHN D. GOULD** is a manager of the Human Factors group at IBM Research. He is interested in human factors and computers. His work involves the development of computer tools and applications that are useful, usable, and well liked. He is wise enough to have worked with both Boies and Lewis.

**STEPHEN J. BOIES** is a senior manager of the ITS project in IBM Research. He is interested in human factors and computers, and has

championed user-interface design research in IBM Research. He makes computer tools and applications that work, and that even Gould can use. **Authors' Present Address:** IBM Research—Hawthorne, P.O. Box 704, Yorktown Heights, NY 10598.

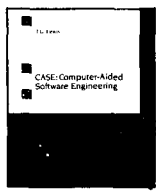
**CLAYTON LEWIS** is an associate professor of computer science at the University of Colorado. He is interested in human factors and computers, sharing his ideas with others through his work, teaching, and communication with colleagues. **Author's Present Address:** Computer Science Department, University of Colorado, Boulder, CO 80309.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1991 ACM 0001-0782/90/1200-074 \$1.50

TO ORDER BY CREDIT CARD CALL TOLL FREE 1-800-926-2665

## HOW CAN YOU SIMPLIFY AND REDUCE THE COST OF SOFTWARE DEVELOPMENT?



You start with

**CASE**

**Computer-Aided  
Software Engineering**

**By T.G. Lewis**


**420 pp., 274 illus., \$45.95**

*Just published!*

This comprehensive guide explains proven ways to apply CASE techniques and tools to write the best possible software. Included are specific examples that take a project from inception to final stage, clearly illustrating how to apply CASE tools to your own applications. The book shows you how and when to use CASE for different functions, including cost estimation, project management, requirement specification, coding, testing and maintenance. And it covers such important new areas as UIMS, object-oriented design, visual programming, and the Spiral Life Cycle model.

To take full advantage of the time- and money-saving benefits of CASE, send for your FREE 15-day examination of **CASE: Computer-Aided Software Engineering** today! No obligation to purchase!

To order by credit card call toll free 1-800-926-2665. Or write to: **VAN NOSTRAND REINHOLD**, Mail Order Dept., P.O. Box 668, Florence, KY 41022-0668.



1/91 C 1129

Circle #38 on Reader Service Card