DESIGN PRINCIPLES FOR HUMAN-COMPUTER INTERFACES

Donald A. Norman
Department of Psychology
and
Institute for Cognitive Science C-015
University of California, San Diego
La Jolla, California 92093

ABSTRACT

If the field of Human Factors in Computer Systems is to be a success it must develop design principles that are useful, principles that apply across a wide range of technologies. In the first part of this paper I discuss some the properties that useful principles should have. While I am at it, I warn of the dangers of the tar pits and the sirens of technology. We cannot avoid these dangers entirely, for were we to do so, we would fail to cope with the real problems and hazards of the field.

The second part of the paper is intended to illustrate the first part through the example of tradeoff analysis. Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another. Tradeoff analysis provides a quantitative method of assessing tradeoff relations for two attributes x_i and x_j by first determining the *User Satisfaction* function for each, *U(x)*, then showing how $U(x_i)$ trades off against $U(x_j)$. In general, the *User Satisfaction* for a system is given by the weighted sum of the *User Satisfaction* values for the attributes. The analysis is used to examine two different tradeoffs of information versus time and editor workspace versus menu size. Tradeoffs involving command languages versus menu-based systems, choices of names, and handheld computers versus workstations are examined briefly.

If we intend a science of human-computer interaction, it is essential that we have principles from which to derive the manner of the interaction between person and computer. It is easy to devise experiments to test this idea or that, to compare and contrast alternatives, or to evaluate the quality of the latest technological offering. But we must aspire to more than responsiveness to the current need. The technology upon which the human-computer interface is built changes rapidly relative to the time with which psychological experimentation yields answers. If we do not take care, today's answers apply only to yesterday's concerns.

Our design principles must be of sufficient generality that they will outlast the technological demands of the moment. But there is a second and most important criterion: the principles must yield sufficiently precise answers that they can actually be of use: Statements that proclaim "Consider the user" are valid, but worthless. We need more precise principles.

This new field — Human Factors in Computer Systems — contains an unruly mixture of theoretical issues and practical problems. Just as it is important that our theoretical concerns have breadth, generality, and usability, so too is it important that we understand the practical problems. We are blessed with an exciting, rapidly developing technology that is controlled through the time consuming and addictive procedure called programming. There are traps for the unwary: let me tell you about them.

Tar Pits and Sirens of Technology

As with most unexplored territories, dangers await: tar pits and sirens. The former lie hidden in the path, ready to

This research was supported by Contract N00014-79-C-0323, NR 667-437 with the Personnel and Training Research Programs of the Office of Naval Research and by a grant from the System Development Foundation. Requests for reprints should be sent to Donald A. Norman, Institute for Cognitive Science C-015; University of California, San Diego; La Jolla, California, 92093, USA.

These ideas have benefited greatly by interactions with the UCSD Human-Machine Interaction project, especially Liam Bannon, Allen Cypher, Steve Draper, Dave Owen, Mary Riley, and Paul Smolensky. Comments on a draft of the paper by Danny Bobrow, Jonathan Grudin, Peter Jackson, Allen Munro, and Julie Norman resulted in major improvements of the ideas and exposition.

trap the unwary. The latter stand openly, luring their prey to destruction with bewitching sweetness. I see too many of you trapped by one or the other.

To program or not to program, that is the question. Whether it is nobler to build systems or to remain pure, arguing for abstract principles independent of the technology. Build systems and you face the tar pits, writing programs whose sole justification is to support the writing of programs, eating up work-years, eating up resources, forever making "one last improvement." When you finish, others may look and nod, saying, "yes, how clever." But will anything general be learned? Will the next technological leap pass it by? Programming can be a pit that grabs the unwary and holds them down. While in the pit they may struggle and attract attention. Afterwards, there may be no visible trace of their passing.

Alternatively, you may be seduced by the sirens of technology. High resolution screens, color, three-dimensions, mice, eye-movement detectors, voice-in, voice-out, touch-in, feelers-out; you name it, it will happen. Superficial pleasure, but not necessarily any lasting result. What general lessons will have been learned?

Damned if you do, damned if you don't. The pure in heart will avoid the struggles, detour the tar pits, blind their eyes to the sirens. "We want general principles that are independent of technology," they proclaim. But then what should they study? If the studies are truly independent of the technology, they are apt to have little applicability. How can you develop useful principles unless you understand the powers and weaknesses of the technology, the pressures and constraints of real design? Study a general problem such as the choice of editor commands and someone will develop a new philosophy of editing, or a new technological device that makes the old work irrelevant. The problem is that in avoiding the paths that contain the tar, you may never reach any destination; in avoiding temptation, you remain pure, but irrelevant. Life is tar pits and sirens. Real design of real systems is filled with the messy constraints of life: time pressures, budget limitations, a lack of information, abilities, and energy. We are apt not to be useful unless we understand these constraints and provide tools that can succeed despite them, or better, that can help alleviate them. Experimental psychology is not noted for its contributions to life; the study of human-computer interface should be.

Four Strategies for Providing Design Principles

What can we accomplish? One thing that is needed is a way of introducing good design principles into the design stage. How can we do this? Let me mention four ways.

1: Try to impress upon the designer the seriousness of the matter, to develop an awareness that users of systems have special needs that must be taken account of. The problem with this approach is that although such awareness is essential, good intentions do not necessarily lead to good design. Designers need to know what to do and how to do it.

2: Provide methods and guidelines. Quantitative methods are better than qualitative ones, but all are better than none at all. These methods and guidelines must be usable, they must be justifiable, they must have face validity. The designer is apt to be suspicious of many of our intentions. Moreover, unless we have worked out these guidelines with skill, they will be useless when confronted with the realities of design pressures. The rules must not only be justified by reasonable criteria, they must also appear to be reasonable: designers are not apt to care about the discussions in the theoretical journals.

3: Provide software tools for interface design. This can be a major positive force. Consider the problem of enforcing consistent procedures across all components of a system. With appropriate software tools, consistency can be enforced, if only because it will be easier to use the tools rather than to do without them. We can ensure reasonable design by building the principles into the tools.

4: Separate the interface design from other programming tasks. Make the interface a separate data module, communicating with programs and the operating system through a standardized communication channel and language. Interface design should be its own discipline, for it requires sophistication in both programming and human behavior. If we had the proper modularization, then the interface designer could modify the interface independently of the rest of the system. Similarly, many system changes would not require modification of the interface. The ideal method would be for software tools to be developed that can be used in the interface design by non-programmers. I imagine the day when I can self-tailor my own interface, carrying the specification around on a micro-chip embedded in a plastic card. Walk up to any computer terminal in the world, insert my card, and *voila*, it is my personalized terminal.

I recommend that we move toward all of these things. I have ordered the list in terms of my preferences: last being most favored; first being easiest and most likely today. Each is difficult, each requires work.

There has been progress towards the development of appropriate design methods. One approach is demonstrated through the work of Card, Moran, and Newell (1983) who developed formal quantitative methods of assessing a design. Their techniques provide tools for the second of my suggested procedures. Card, Moran, and Newell emphasize the micro-processes of interaction with a computer — for example, analysis at the level of keystrokes. At UCSD, we are attempting to develop other procedures. In the end, the field will need many methods and guidelines, each complementing and supplementing the others. Let me now describe briefly the approach that we are following, then present one of our techniques — the tradeoff analysis — in detail.

The UCSD User Centered System Design Project

At UCSD we have a large and active group attempting to put our philosophy into practice. Our goal is to have pure heart and clear mind, even while feet and loins are in tar and temptation. Some of our initial activities are being presented in this conference: Bannon, Cypher, Greenspan, and Monty (1983); O'Malley, Smolensky, Bannon, Conway, Graham, Sokolov, and Monty (1983); Root and Draper (1983). Other examples have been published elsewhere or are still undergoing final development (Norman, 1983a, b, c).

The principles that we follow take the form of statements plus elaboration, the statements becoming slogans that guide the research. The primary principle is summarized by the slogan that has become the name of the project: *User Centered System Design.* The slogan emphasizes our belief that to develop design principles relevant to building human-machine interfaces, it is necessary to focus on the user of the system. This focus leads us naturally to a set of topics and methods. It means we must observe how people make use of computer systems. It brings to the fore the study of the *mental models* that users form of the systems with which they interact. This, in turn, leads to three related concepts: the designer's view of the system — the *conceptual model*; the image that the system presents to the user — the *system image*; and third, the *mental model* the user develops of the system, mediated to a large extent by the system image. We believe that it is the task of the designer to establish a conceptual image of the system that is appropriate for the task and the class of users, then to construct the system so that the system image guides the user to acquire a mental model that matches the designer's conceptual model.

The Slogans

There are five major slogans that guide the work:

- There are no simple answers, only tradeoffs.
- There are no errors: all operations are iterations towards a goal.
- Low level protocols are critical.
- Activities are structured.
- Information retrieval dominates activity.

There are no simple answers, only tradeoffs. A central theme of our work is that, in design, there are no correct answers, only tradeoffs. Each application of a design principle has its strengths and weaknesses; each principle must be interpreted in a context. One of our goals is to make the tradeoffs explicit. This point will be the topic of the second half of the paper.

All operations are iterations towards a goal. A second theme is that all actions of users should be considered as part of their attempt to accomplish their goals. Thus, even when there is an error, it should be viewed as an attempt by the user to get to the goal. Typing mistakes or illegal statements can be thought of as an approximation. The task for the designer, then, is to consider each input as a starting point and to provide appropriate assistance to allow efficient modification. In this way, we aid the user in rapid convergence to the desired goal. An important implication of this philosophy is that the users' intentions be knowable. In some cases we believe this can be done by having the users state intentions explicitly. Because many commands confound intentions and actions, intentions may substitute for commands.

Low level protocols are critical. By "protocol," we mean the procedures to be followed during the conduct of a particular action or session, this meaning being derived from the traditional meaning of protocol as "a code of diplomatic or military etiquette or precedence." Low level protocols refer to the actual operations performed by the user — button pushes, keypresses, or mouse operation — and these permeate the entire use of the system. If these protocols can be made consistent, then a major standardization takes place across all systems.

Activities are structured. User actions have an implicit grouping corresponding to user goals; these goals may be interrelated in various ways. Thus, a subgoal of a task is related to the main task in a different way than is a diversion, although both may require temporary cessation of the main task, the starting up of new tasks, and eventual return to the main one. We believe the grouping of user goals should be made explicit, both to the user and to the system, and that doing so will provide many opportunities for improved management of the interaction. For example, the system could constrain interpretation of user inputs by the context defined by the current activity, the system could remind users of where they are as they progress through a collection of tasks, or, upon request, it could provide suggestions of how to accomplish the current task by suggesting possible sequences of actions. The philosophy is to structure activities and actions so that the users perceive themselves as selecting among a set of related, structured operations, with the set understood and supported intelligently by the system (see Bannon, Cypher, Greenspan, & Monty, 1983).

Information retrieval dominates activity. Using a computer system involves stages of activities that include forming an intention, choosing an action, specifying that action to the system, and evaluating the outcome. These activities depend heavily upon the strengths and weaknesses of human short- and long-term memory. This means that we place emphasis upon appropriate design of file and directory structures, command "workbenches," and the ability to get information, instruction, and help on the different aspects of the system. We are studying various representational structures, including semantic networks, schema structures of both conventional and "additive memory" form, browsers, hyper-text structures, and other retrieval aids (see O'Malley, Smolensky, Bannon, Conway, Graham, Sokolov & Monty, 1983).

A Demonstration System

This is where we traverse the tar pits. We feel it essential that our ideas be tested within a working system, not only because we feel that the real constraints of developing a full, usable system are important design considerations that must be faced, but also because we believe that full evaluation can only take place within the bounds of a complete, working environment. Therefore, we intend to construct a test and demonstration system based around a modern workstation using the UNIX operating system. UNIX was chosen because it provides a rich, powerful operating environment. However, because UNIX was designed for the professional programmer, unsophisticated users have great trouble with it, providing a rich set of opportunities for our research.

Although we intend that our design principles will be applicable to any system regardless of the particular hardware being used, many of the concepts are effective only on high-resolution displays that allow multiple windows on the screen and that use simple pointing devices. These displays allow for a considerable improvement in the design of human-computer interfaces. We see no choice but to brave the sirens of technology. The capabilities of the hardware factor into the tradeoff relationships. We intend the demonstration systems to show how the tradeoffs in design choices interact with the technology.

Tradeoffs in Design

Now let us examine one of our proposals — tradeoffs — as a prototype of a quantitative design rule. It is well known that different tasks and classes of users have different needs and requirements. No single interface method can satisfy all. Any single design technique is apt to have its virtues along one dimension compensated by deficiencies along another. Each technique provides a set of tradeoffs. The design choices depend upon the technology being used, the class of users, the goals of the design, and which aspects of interface should gain, which should lose. This focus on the tradeoffs emphasizes that the design problem must be looked at as a whole, not in isolated pieces, for the optimal choice for one part of the problem will probably not be optimal for another. According to this view, there are no correct answers, only tradeoffs among alternatives.

The Prototypical Tradeoff: Information Versus Time

One basic tradeoff pervades many design issues:

Factors that increase informativeness tend to decrease the amount of available workspace and system responsiveness.

On the one hand, the more informative and complete the display, the more useful when the user has doubts or lacks understanding. On the other hand, the more complete the display, the longer it takes to be displayed and the more space it must occupy physically. This tradeoff of amount of information versus space and time appears in many guises and is one of the major interface issues that must be handled. To appreciate its importance, one has only to examine a few recent commercial offerings, highly touted for their innovative (and impressive) human factors design that were intended to make the system easy and pleasurable to use, but which so degraded system response time that serious user complaints resulted.

It is often stated that current computer systems do not provide beginning users with sufficient information. However, the long, informative displays or sequence of questions, options, or menus that may make a system usable by the beginner are disruptive to the expert who knows exactly what action is to be specified and wishes to minimize the time and mental effort required to do the specification. We pit the expert's requirement for ease of specification against the beginner's requirement for knowledge.

I approach this problem by tackling the following questions:

- How can we specify the gain in user satisfaction that results from increasing the size of a menu;
- How do we specify the user satisfaction for the size of the workspace in a text editor;
- How can we specify the loss in user satisfaction from the increase in time to generate the display and decrease in available workspace;
- How can we select menu size, workspace, and response time, when each variable affects the others?

I propose that we answer the question by use of a psychological measure of *User Satisfaction.* This allows us to determine the impact of changing *physical* parameters upon the *psychological* variable of user satisfaction. Once we know how each dimension of choice affects user satisfaction, then we can directly assess the tradeoffs among the dimensions.

Example: Menu Size and Display Time

Let $U(x)$, the user satisfaction for attribute x, be given by a power function, $U(x) = kx^p$. (In Norman, 1983c, I give more details of the method. See Stevens, 1974, for a review of the power function in Psychology.) For the examples in this paper I used the method of magnitude production to estimate parameters.

User preference for menu size. The preferred amount of information must vary with the task, but informal experiments with a variety of menus and tasks suggest that for many situations, about 300 characters is reasonable: I assigned it a satisfaction value of 50. This is the size menu that can be requested for our laboratory's computer mail program ("msg"). It serves as a reminder for 26 single-letter mnemonic commands. To do the power function estimates,

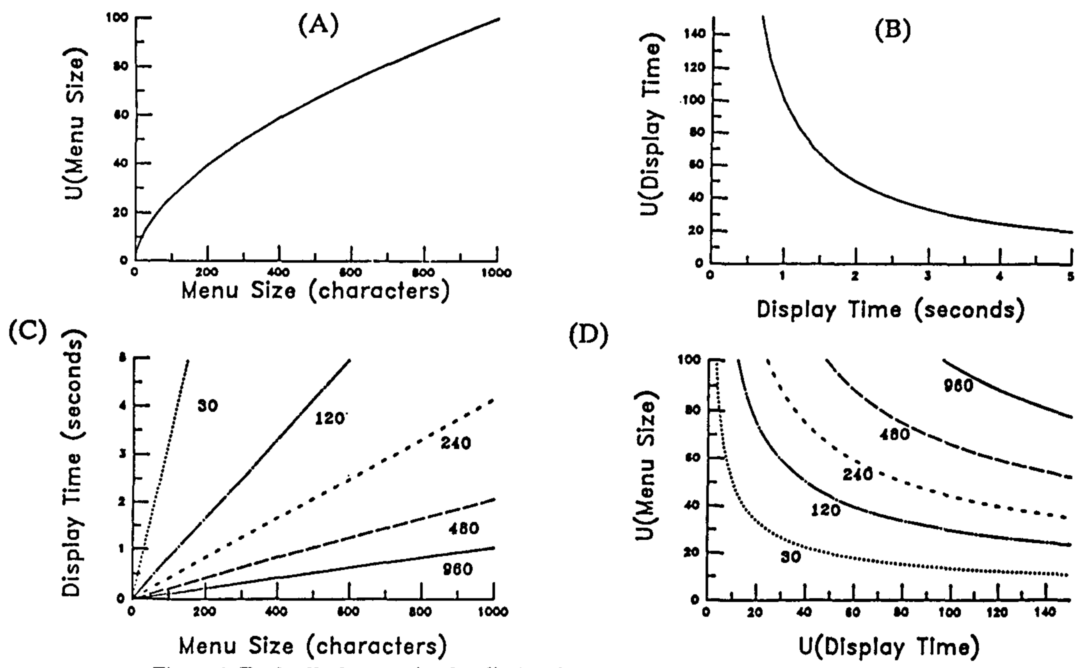

Figure 1. Tradeoff of menu size for display time. Panels A and B show User Satisfaction for menu size, $U(S) = 1.9S^{0.6}$, and display time, $U(T) = 100T^{-1}$, respectively. Panel C shows display time as a function of menu size, $T = S/\beta$, for different values of display rate, β (specified in *characters/second*). Panel D shows the tradeoff between $U(S)$ and $U(T)$ for different values of display rate (β).

I examined a variety of menus of different sizes for the message system (thereby keeping the task the same). I estimated that the menu size would have to increase to half the normal video terminal screen (1000 characters) in order to double my satisfaction. This is a typical result of psychological scaling; a substantial increase of the current value is required to make the increase worthwhile. If $U(300) = 50$ and $U(1000) = 100$, then the parameters of the power function are $k = 1.9$ and $p = 0.6$: $U(S) = 1.9S^{0.6}$.

User preference for response time. There already exists some literature on user satisfaction for response time: the judgements of "acceptable" response times given by Shneiderman (1980, p 228: the times are taken from Miller, 1968). The times depend upon the task being performed. For highly interactive tasks, where the system has just changed state and the users are about to do a new action, 2.0 seconds seems appropriate. I determined that I would be twice as satisfied with a response time of 1 second. Therefore, $U(2\,sec) = 50$ and $U(1\,sec) = 100$. For these values, the power function becomes $U(T) = 100/T$ ($k = 100$, $= -1$).

Size of menu and display time. We need one more thing to complete the tradeoff analysis: the relationship between menu size (S) and the time to present the information (T). In general, time to present a display is a linear function of $S: T = \sigma + S/\beta$, where S is measured in characters, β is the display rate in characters per second (cps) and σ is system response time.

The tradeoff of menu size for display time. Knowledge of $U(S)$, $U(T)$, and the relationship between S and T, lets us determine the tradeoff between User Satisfaction for size of the menu and for time to display the information: $U(S)$ versus $U(T)$. If $\sigma << S/\beta$, then we can probably ignore σ, letting $T = S/\beta$. This lets us solve the tradeoff exactly. [1] If the two power functions are given by $U(S) = aS^p$ and $U(T) = bT^q$, then $U(S) = kU(T)^{p/q}$, where $k = \frac{a}{b^{p/q}}\beta^p$. The tradeoff relationship using the parameters estimated for menus is shown in Figure 1.

Maximizing Total User Satisfaction

Let overall satisfaction for the system, $U(system)$, be given by the weighted sum of the $U(x_i)$ values for each of its attributes, x_i: $U(system) = \sum \omega_i U(x_i)$, where ω_i is the weight for the *i-th* attribute. When there are only two attributes, x_1 and x_2, if we hold $U(system)$ constant at some value C, we can determine the *iso-satisfaction* line: $U(x_1) = \frac{C}{\omega_1} - \frac{\omega_2}{\omega_1}U(x_2)$. Thus, the iso-satisfaction functions appear on the tradeoff graphs as straight lines with a slope of $-\omega_2/\omega_1$, with higher lines representing higher values of $U(system)$.

If the tradeoff functions are concave downward (as are some in later figures), the maximum satisfaction occurs where the slope of the tradeoff function is tangent to the iso-satisfaction function: that is, when the slope of the tradeoff function $= -\omega_2/\omega_1$. In this case, maximum satisfaction occurs at some compromise between the two variables.

---

1. Letting $\sigma = 0$ simplifies the tradeoff relations, but this is not a necessary assumption. If system response time is slow, then σ should be reinstated: the tradeoff can still be be determined quite simply.

If the tradeoff functions are concave upward (as in Figure 1), then the *minimum* satisfaction occurs where the iso-satisfaction curves are tangent to the tradeoff functions. Maximum satisfaction occurs by maximizing one of the two attributes. The expert, for whom $\omega_T/\omega_S >> 1$, will not sacrifice time for a menu. The beginner, for whom $\omega_T/\omega_S << 1$, will sacrifice display time in order to get as big a menu as possible. For intermediate cases between that of the extreme expert or beginner, the optimum solution is still either to maximize menu size or to minimize display time, but the user might be indifferent as to which of these two was preferred. These conclusions apply to the tradeoff functions of Figure 1D regardless of display rate, as long as the curves are concave upward.[2]

These analyses say that the tradeoff solution that one tends to think of first — to compromise between time and menu size by presenting a small menu at some medium amount of workspace — actually provides the least amount of total satisfaction. Satisfaction is maximized by an all or none solution. The all-or-none preference applies only to tradeoff functions that are concave upward, such as that between menu size and display time. Later we shall see that when time is not relevant, the analysis of the tradeoff between menu size and workspace predicts that even experts will sacrifice some workspace for a menu.

Workspace

Available workspace refers to the amount of room left on the screen after the menu (or other information) is displayed. This is especially important where the menu stays on the screen while normal work continues. The tradeoff is sensitive to screen size. If we had a screen which could display 60 lines of text, using 6 lines to show the current state of the system and a small menu of choices would not decrease usability much. But if the screen could only display 8 lines at a time, then using 6 of them for this purpose would be quite detrimental.

User preference function for workspace. The user preference function for workspace clearly depends upon the nature of the task: some tasks — such as issuing a command — may only require a workspace of 1 line, others — such as file or text editing — could use unlimited workspace. Let us consider the workspace preferences for text editing of manuscripts. The most common editors can only show 24 lines, each of 80 characters: 1920 character positions. I let $U(1920) = 50$. To estimate the workspace that would double the value, I imagined working with screen editors of the sizes shown in Table 1. I concluded that I would need the size given by the two page journal spread. That is, $U(6400) = 100$. The power function parameters are $k = 0.64$ and $p = 0.6$, so that $U(w) = 0.64w^{0.6}$: the same exponent used for menu size but with a different scale factor, k. This function is shown in Figure 2.

2. Whether a tradeoff function exhibits upward or downward concavity depends upon the choice of user satisfaction function. If both functions are power functions with one exponent positive and the other negative, the tradeoff functions are always concave upward. When both exponents are positive, the tradeoff functions are always concave downward. When the two functions are logarithmic, the tradeoff functions are always concave downward, and when they are both logistic, the tradeoff functions are both concave upward and downward, switching from one to the other as a function of the other variables (e.g., display rate). These conclusions hold whenever the two variables, x_1 and x_2, are linearly related.

Table 1

SIZE OF COMMON TEXTS AND DEVICES (in characters)	
TEXT OR DEVICE	APPROXIMATE NUMBER OF CHARACTER POSITIONS
Portable Computer (*Radio Shack Model 100*)	320
Home Microprocessor (*Apple II*)	960
Standard Video Display Unit	1,920
One typed manuscript page (*double spaced*)	2,600
One typed manuscript page (*single spaced*)	4,000
Journal page (*Cognitive Science*)	3,200
Double page spread	6,400
Page of Proceedings (*Gaithersberg Human Factors in Computer Systems*)	5,500
Double page spread	11,000
Newspaper page (*Los Angeles Times*)	30,000
Double page spread	60,000

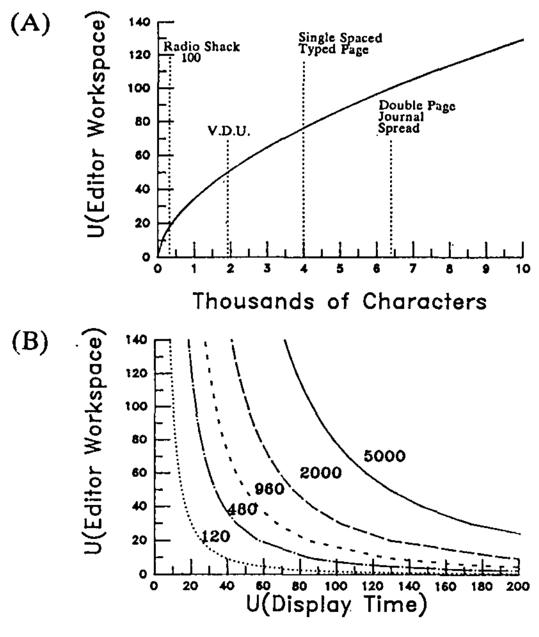

Figure 2. Panel A. User Satisfaction function for editor workspace, w: $U(w) = 0.64w^{0.6}$. Typical character sizes for various displays and texts are also shown. Panel B shows the tradeoff function of workspace against time, $U(w)$ versus $U(T)$, for different display rates, β (*characters/second*). $U(T)$ is shown in Figure 1B: $U(T) = 100T^{-1}$.

Trading workspace for display time. One penalty for increasing the size of the workspace is increased time to display the workspace. If we use the same User Satisfaction function for time as in Figure 1B, we get the tradeoff functions shown in Figure 2B. Large workspaces require very high display rates before they are satisfactory. Because these tradeoff functions are concave upward (and are very similar to the functions of Figure 1D), the same conclusions apply here as to those earlier functions: the optimum operating point is an all-or-none solution. Thus, the user either prefers a large workspace, regardless of the time penalty, or a very fast display, regardless of the workspace penalty. Here, however, the relative weights are apt to be determined by the task rather than by the user's level of skill.

Suppose the task were one in which the display changes relatively infrequently. In this case we would expect $\omega_{workspace} >> \omega_{display\ time}$, so that the optimum solution is to have as big a workspace as possible. If the task were one that requires frequent changes in the display, then we would expect the reverse result: $\omega_{workspace} << \omega_{display\ time}$, so the optimum solution is to shrink the workspace to the smallest size at which the task can still be carried out, thereby minimizing display time.

Trading workspace for menu size. Adding a menu to the display decreases the amount of available workspace. Let W be the total size of the workspace that is available for use, w the workspace allocated to the text editor, and m the space allocated for a menu: $w = W - m$. We know that $U(m) = am^p$ and $U(w) = b(W-m)^q$, where $a = 2$, $b = 0.6$, and $p = q = 0.6$. This leads to the tradeoff functions shown in Figure 3.

Note that $U(editor\ workspace)$ is relatively insensitive to $U(menu\ size)$. This is because a relatively small sized display makes a satisfactory menu, whereas it requires a large display to make a satisfactory editor workspace. As a result, changing the size of the menu by only a few lines can make a large change in User Satisfaction, whereas the same change in workspace is usually of little consequence.

In some commercially available editors, the menu of commands can occupy approximately half the screen (usually 24 lines). Figure 1 indicates that for a menu of 12 lines (960 characters), $U(menu) \approx 100$. However, from Figures 2 and 3 we see that with a workspace of only 1920 characters, a menu of around 1000 characters (or of $U(menu) = 100$) reduces $U(editor\ workspace)$ from its value of 50 with no menu to 34: a reduction of almost one third. From Figure 3 we see that we would be much less impaired by the same size menu were the workspace considerably greater. In such cases, we have a clear tradeoff between the need for the menu information and the desire to have a reasonable workspace.

Maximizing total user satisfaction for menu and workspace. When tradeoff functions are concave downward (as in Figure 3), maximum satisfaction occurs where the slope of the tradeoff function is tangent to the iso-satisfaction function. For the user who values workspace and menu equally (so that the preferred slope is -1), the optimum solution is to operate at the right hand side of Figure 3. This makes for a relatively high value of user satisfaction for the menu (which means a large menu — the exact sizes can be determined from Figure 1A) — but with little sacrifice in user satisfaction for workspace. The more expert user will have an iso-satisfaction function with a much smaller slope, and so will sacrifice menu for workspace. Similarly, the beginner will have an iso-satisfaction curve with high slope which will maximize menu size at the expense of workspace. These results are quite unlike the tradeoffs that involved time in which an all-or-none solution was optimal: here, the optimum values are compromises between workspace and editor size. Display rate and amount of total available workspace alter the point of optimum operation. The analysis provides exact numerical determination of how the optimal operating point is affected by these variables.

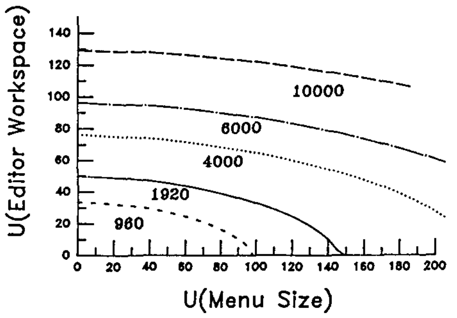

Figure 3. The tradeoff of User Satisfaction for menu size against User Satisfaction for editor workspace, for different values of total workspace, W. Horizontal lines represent constant values of $U(m)$ and, therefore, of m: the values of m can be determined from panel A of Figure 1. Similarly, vertical lines represent constant values of w: the values of w can be determined from panel A of Figure 2.

A Critique of the Tradeoff Analysis

There are a number of problems with the tradeoff analyses presented in this paper. There are two major criticisms, one minor one. Let me start with the minor one, for it represents a misunderstanding that would be good to clear up. I illustrate the misunderstanding for the variable of menu size, but the discussion applies to other variables as well:

- The functions must be wrong: *U(menu size)* continually increases as a function of *menu size*, yet when the size gets too large, the menu becomes less useful: *U(menu size)* should also decrease.

User Satisfaction for the System Is the Sum of Its Parts

This misunderstanding derives from confusing *User Satisfaction* for a single attribute with *User Satisfaction* for a system. A major philosophy of the tradeoffs analysis is that a system can be decomposed into its underlying component attributes and *User Satisfaction* for each assessed individually. The *User Satisfaction* for the entire system can only be determined from the combination of the *User Satisfaction* values for each of its components. The satisfaction for the amount of information conveyed by the menu continues to increase with size, but that the ability to find something (captured by "search time") decreases with increasing size of a menu. The overall satisfaction for the menu is given by the sum of the increasing satisfaction for the information and the decreasing satisfaction for search effort: the result is a U-shaped curve that decreases as size gets too big.

Now let me address the two major critiques:

- The tradeoff functions are arbitrary;
- How do we determine the functions when we design? It would be more useful were there a set of standards (perhaps in handbook form);

These two issues point to unsolved problems with the method. My defense is to argue that this procedure is new. The goal is to introduce the philosophy and to encourage others to help in the collection of the relevant data and in the development of the method. However, the numbers and the particular functions used here may be useful, for the tasks for which they were derived: they do mesh well with my intuitions.

The Tradeoff Functions Are Arbitrary

Although the functions used here are indeed arbitrary, three things need to be noted. First, power functions have a long tradition of satisfactory use in psychology and so are apt to be good approximations. Second, I have actually computed User Satisfaction functions using the logistic, power, and logarithmic functions; over much of the range of interest, the results differed suprisingly little, although at high data rates, the concave upward tradeoff functions became concave downward when the logistic was used for size and time, although at low data rates they were still concave upward (see footnote 2). Third, I agree that the preferred thing would be to have an experimental program to determine the exact forms and parameters of the functions. In particular, the all-or-none prediction is sensitive to the form of the User Satisfaction functions.

How Do We Determine the Functions When We Design?

Here, again, empirical work is needed. I suspect the functions will be found to vary only for a reasonably small number of classes of users, classes of tasks, and design attributes, so that it would be possible to collect typical values in a handbook. Alternatively, quick data collection methods might be devised: the magnitude estimation procedures are especially easy to apply. A handbook might be quite valuable. Before this can be done, of course, it is necessary to determine that the hypothesis is correct — that there are a relatively small number of tasks, user classes, and tradeoff variables that need be considered. Moreover, one must extend the analysis to a larger domain of problems and demonstrate its usefulness in actual design.

Some Other Examples of Tradeoffs

There are numerous other tradeoff analyses in addition to the ones presented here. Three other situations seem important enough to warrant consideration here, even though they are not yet ready for quantitative treatment. These are: (1) the comparison between command languages and menu-driven systems; (2) how to choose names for commands and files; and (3), the tradeoffs that result when moving among computer systems of widely varying capabilities, as in the differences between hand-held computers and powerful, networked workstations.

Command Languages Versus Menus

The relative merits of menu-based systems and command language systems are often debated, seldom with any firm conclusion. It is useful to compare their tradeoffs, but before we do, it is necessary to be clear about what is meant by each alternative. In this context a command language system is one in which no aids are presented to the user during the intention or choice stages, and the action specification is performed by typing a command, using the syntax required by the operating system. (The distinctions among the intention, choice, and specification stages come from Norman, 1983b.) Command languages are the most frequent method of implementing operating systems. Similarly, in this context a menu-based system is one in which all commands are presented via menus, where a command cannot be specified unless it is currently being shown on the active menu, and where the commands are specified either through short names or single characters (as indicated by the menu items) or by pointing at the relevant menu item (or perhaps at a switch or mark indicated by the item). These are restricted interpretations of the two alternatives, confounding issues about the format for information presentation and action specification. Still, because they represent common design alternatives, it is useful to compare them.

Command languages offer experts great versatility. Because of their large amount of knowledge and experience with the system, experts tend to know exactly what operations they wish performed. With a command language they can specify their operations directly simply by typing the names of the commands, as well as any parameters, files, or other system options that are required. Command languages make it easy to specify parameters (or "flags") to commands, something that may be difficult with menu-based systems.

Menus offer the beginner or the casual user considerable assistance. At any stage, information is available. Even abbreviated menus serve as a reminder of the alternatives. Experts often complain about menu-based systems because of the time penalty in requesting a menu, waiting for it to be displayed, and then searching for the desired item. Moreover, systems with large numbers of commands require multiple menus that slow up the expert. The problem is that the system is designed to give help, whether or not the user wishes it.

Two of the difficulties with menus are the delay in waiting for them to be plotted and the amount of space they occupy. Figure 1D shows that the tradeoff between amount of information and time delay is especially sensitive to information transmission rate. When transmission time becomes fast enough, there is little penalty for menus, whereas at slow rates of data transmission, the penalty is high. In similar fashion, Figure 3 shows that the tradeoff between menu size and workspace is especially sensitive to the amount of total workspace available. When sufficient workspace is available, there is little penalty for menus. Thus, slow transmission rates and small workspaces bias the design choice toward command language systems; high data rates and large workspaces bias the system toward menu-based systems.

The two systems also differ in the kinds of errors they lead to and ease of error correction. In a command language system, an error in command specification usually leads to an illegal command: no action gets performed. This error is usually easy to detect and to correct. In a menu-based system, an error in specification is almost always a legal command. This error can be very difficult to correct. If the action was subtle, the user may not even be aware it was performed. If the action was dramatic, the user will often have no idea of what precipitated it, since the action specification was unintentional.

Some of the tradeoffs associated with menu-based systems and command language systems are summarized in Table 2. Command languages tend to be virtuous for the expert, but difficult for the novice; they are difficult to learn and there are no on-line reminders of the set of possible actions. Menus are easy to use and they provide a constant reminder. On the other hand, menus tend to be slow — for some purposes, the expert finds them tedious and unwieldy — and not as flexible as command languages.

This analysis is brief and restricted to the particular formats of command language and menu-based systems that were described. There do exist techniques for mitigating the deficiencies of each system. Nonetheless, the analysis is useful, both for pointing out the nature of the issues and for being reasonably faithful to some existing systems. In the argument over which system is best, the answer must be that neither is: each has its virtues and its deficiencies.

The Choice of Names for Commands and Files

Another example of a common tradeoff is in the choice of name for a command or a file. The problem occurs because the name must serve two different purposes: as a *description* of the item and also as the string of characters that must be typed to invoke it, that is, as the *specification*; these two uses pose conflicting requirements.

Consider the properties of names when used as descriptions. The more complete the description, the more useful it can be, especially when the user is unsure of the options or is selecting from an unfamiliar set of alternatives. However, the longer the description, the more space it occupies and the more difficult to read or scan the material. In addition, there are often system limitations on the length and format of names. For these reasons, one usually settles for a partial description, counting on context or prior knowledge to allow the full description to be regenerated by the user.

Table 2

TRADEOFFS BETWEEN MENU-BASED SYSTEMS AND COMMAND LANGUAGE SYSTEMS

ATTRIBUTE	MENU-BASED	COMMAND LANGUAGE
Speed of use:	Slow, especially if large or if has hierarchical structure.	Fast, for experts; operation can be specified exactly, regardless of system state.
Prior knowledge required:	Very little — can be self-explanatory.	Considerable — user is expected to have learned set of alternative actions and command language that specifies them.
Ease of learning:	High. Uses recognition memory: easier and more accurate than recall memory. Easy to explore system and discover options.	Low. Users must learn names and syntax of language. If alternatives are numerous, learning may take considerable time. No simple way to explore system and discover options not already known.
Errors:	Specification error leads to inappropriate action: difficult to determine what happened and to correct.	Specification error usually leads to illegal command: easy to detect, easy to correct.
Most useful for:	Beginner or infrequent user.	Expert or frequent user.

Once the appropriate name has been determined, the user enters the specification stage of operation; the user must specify to the computer system which name is desired. Most users are not expert typists, and so it is desirable to simplify the specification stage. As a result, there is pressure toward the use of short names, oftentimes to the limit of single character command names. [3]

The desirability for short names is primarily a factor when specification must be done by naming. When the specification can be done by pointing, then ease of typing is no longer a factor. Nonetheless, there are still constraints on the name choice: the longer the name, the easier to find and point at the desired item, but at the cost of using a larger percentage of the available workspace, of increasing display time, and the ease of reading and search. Now names might wish to be chosen so they are visually distinct, or so that they occupy appropriate spatial locations on the display, in all cases adding more constraints to the naming

----

3. A number of systems allow for shortcuts in specification, so that one need only use sufficient characters (plus some "escape" or "wild-card" character) to make the name unique. This option poses its own naming constraints; now a name is chosen not only to be descriptive, but with the added requirement that one or two letters be sufficient to distinguish it uniquely from all other names. The typing aid introduces its own form of naming constraint.

problem. In general the descriptive requirements tend to push toward longer names, names that provide as much information as possible. The specification requirements tend to push toward shorter names, names that are easy to type.

Handheld Computers Versus Workstations

New developments in technology are moving computer systems in several conflicting directions simultaneously. Workstations are getting more powerful, with large memories, large, high resolution screens, and with very high communication bandwidths. These developments move us toward the ability to present as much information as is needed by the user with little penalty in time, workspace, or even memory space. At the same time, some machines are getting smaller, providing us with briefcase sized and handheld computers. These machines have great virtue because of their portability, but severe limitations in communications speed, memory capacity, and amount of display screen or workspace.

Just as workstations are starting to move toward displays capable of 1000 line resolution, showing several entire pages of text, handheld computers move us back toward only a few short lines — perhaps 8 lines of 40 characters each — and communication rates of 30 cps (300 baud). The major differences between workstations and handheld computers relevant to the tradeoffs discussion are in the amount of memory, processor speed and power, communication abilities, availability of extra peripherals, and screen size: in all cases, the handheld machine has sacrificed power for portability. Because the same people may wish to use both handheld machines and workstations (one while at home or travelling, the other at work), the person may wish the same programs to operate on the two machines. However, the interface design must be different, as the tradeoff analyses of this paper show.

Summary and Conclusions

The tradeoff analysis is intended to serve as an example of a quantitative design tool. In some cases it may not be possible to select an optimum design, not even for a restricted class of activities and users. In these cases, knowledge of the tradeoffs allows the designer to choose intelligently, knowing exactly what benefits and limits the system design will provide. Finally, the analyses show that some design decisions are heavily affected by technology, others are not. Thus, answers to design questions are heavily context dependent, being affected by the classes of users for whom the system is intended, the types of applications being performed, which stages of user activities are thought to be of most importance, and the level of technology being employed.

The work presented here is just the beginning. In the ideal case, the tradeoff relationships will be known exactly, perhaps with the relevant quantitative parameters provided in handbooks. This paper has limited itself to demonstrating the basic principles. Considerable development must still be done on this issue and on the other major parameters and issues that affect the quality of the human-machine interaction. Much work remains to be done.

A second point of the paper is to argue for more fundamental approaches to the study of human-machine interaction. All too often we are presented with minor studies that do not lead to general application, or to studies that are restricted to a particular technology. All too often we are trapped in the tar pits of the field or seduced by the sirens of technology. If we are to have a science of design that can be of use beyond today's local problems, we must learn to broaden our views, sharpen our methods, and avoid temptation.

A major moral of this paper is that it is essential to analyze separately the different aspects of human-computer interaction. Detailed analyses of each aspect of the human-computer interface are essential, of course, but because design decisions interact across stages and classes of users, we must also develop tools that allow us to ask for what purpose the system is to be used, then to determine how best to accomplish that goal. Only after the global decisions have been made should the details of the interface design be determined.

References

Bannon, L., Cypher, A., Greenspan, S., & Monty, M.L. Evaluation and analysis of users' activity organization. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems.* Boston, December, 1983.

Card, S., Moran, T., & Newell, A., *Applied Information-Processing Psychology: The Human-Computer* Interface. Hillsdale, N.J.: Erlbaum Associates, 1983.

Miller, R. B. Response time in man-computer conversational transactions, *Proceedings of the Spring Joint Computer Conference*, 1968, *33*. Montvale, New Jersey, pp. 267-277.

Norman, D. A. Design rules based on analyses of human error. *Communications of the ACM*, 1983a, *4*, 254-258.

Norman, D. A. Four stages of user activities. Manuscript. 1983b.

Norman, D. A. Tradeoffs in the design of human-computer interfaces. Manuscript. 1983c.

O'Malley, C., Smolensky, P., Bannon, L., Conway, E., Graham, J., Sokolov, J., & Monty, M. L. A proposal for user centered system documentation. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems.* Boston, December, 1983.

Root, R. W., & Draper, S. Questionnaires as a software evaluation tool. *Proceedings of the CHI 1983 Conference on Human Factors in Computer Systems.* Boston, December, 1983.

Shneiderman, B. *Software Psychology: Human Factors in Computer and Information* Systems. Cambridge, Mass.: Winthrop Publishers, 1980.

Stevens, S. S. Perceptual magnitude and its measurement. In E. C. Carterette & M. P. Friedman (Eds.), *Handbook of perception* (Vol. 2). New York: Academic Press, 1974.