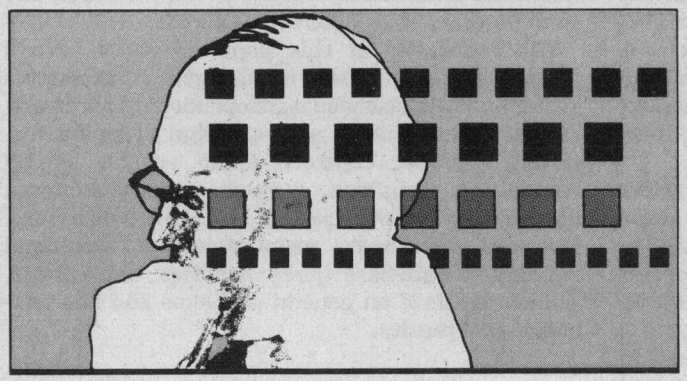
Successful industrial design gracefully unites esthetics and function at minimum cost. However, designers face special problems when they apply their skills to interactive computer systems.



Human Factors Experiments in Designing Interactive Systems

Ben Shneiderman
University of Maryland

Providing useful tools for computer users with a wide range of experience, problems, skills, and expectations is a challenge to scientific competence, engineering ingenuity, and artistic elegance. System developers are increasingly aware that ad hoc design processes, based on intuition and limited experience, may have been adequate for early programming languages and applications but are insufficient for interactive systems which will be used by millions of diverse people. Regular users quickly pass through the gadget fascination stage and become demanding users who expect the system to help them in performance of their work. Clearly, therefore, interactive computer-based consumer products for home, personal, or office applications require increasing levels of design effort.

Unfortunately, it is not possible to offer an algorithm for optimal or even satisfactory design. Interactive system designers, like architects or industrial designers, seek a workable compromise between conflicting design goals. Systems should be simple but powerful, easy to learn but appealing to experienced users, and facilitate error handling but allow freedom of expression. All of this should be accomplished in the shortest possible development time, costs should be kept low, and future modification should be simple. Finding a smooth path through these conflicting goals is a challenge.

Henry Dreyfuss,[1] a leading industrial designer responsible for plane, train, and boat interiors as well as dozens of familiar consumer items, provides useful guidance. He devotes a full chapter to the experience of designing the 500-Type Telephone, the standard rotary dial desk model. Measurements of 2000 human faces were used to determine the spacing between the mouth and ear pieces. After consultation with Bell System engineers about the layout of electronic components, 2500 sketches for possible designs were made. Numerous variations of the handgrip were considered until the familiar rounded-off rectangular cross section was adopted. Variations on dial and faceplate were tested until a 4¼-inch diameter faceplate was selected to replace the older 3-inch version. Placement of the letters and numbers was studied, the angle of the dial was adjusted to reduce glare, and the cradle was modified to minimize the receiver-off-the-hook problem. Accurate layout drawings were made for all the variations, and finally clay and plaster models were built to compare the leading designs. Then testing began.

This process contrasts sharply with most interactive system development experiences where designs are hastily proposed and evaluated informally. Alternative command structures, error handling procedures, or screen formats rarely get implemented for pilot testing purposes. Dreyfuss spends another entire chapter emphasizing the importance of testing. Tests and pilot studies should be more than the informal, biased opinion of a colleague. A pilot test should involve actual users for sufficient time periods to get past initial learning problems and novelty. Conflicting designs should be evaluated in carefully controlled experimental conditions. Though experiments provide no guarantee of quality, they are far better than informal guesswork. The process of developing an experimental comparison can itself be productive, often providing worthwhile insights. Statistical performance data and informal subjective commentary from participants can be valuable in fine-tuning proposed procedures. Experimental research can lead to fundamental insights which transcend specific systems. Nickerson,[2] Bennett,[3] Martin,[4] and Miller and Thomas[5] provide broad-ranging reviews of issues and references for designers and researchers of interactive systems. Shneiderman[6] covers related work in data-base facilities, and other articles in this issue focus on programming language usage.

0018-9162/79/1200-0009$00.75 © 1979 IEEE

Goals for interactive system designers

The diversity of situations in which interactive systems may be used makes it difficult to prescribe a universal set of goals. The attempts of several system designers to define goals are shown in Figures 1 through 8.

Foley and Wallace[15] make their recommendations by enumerating five problem areas: boredom (improper pacing), panic (unexpectedly long delays), frustration (inability to convey intentions or inflexible and unforgiving system), confusion (excessive detail or lack of structure), and discomfort (inappropriate physical environment).

The best detailed guide for design of interactive display systems was developed by Engel and Granda.[16] They make specific suggestions about display formats, frame contents, command language, recovery procedures, user entry techniques, general principles, and response time requirements.

Unfortunately, these lists are only crude guides to the designer. The entries are not independent and sometimes are in conflict. The lists contain contradictory recommendations and are certainly incomplete. Finally, these design goals are largely unmeasurable. Can we assign a numerical value to the simplicity, stability, responsiveness, variety, etc., of a system? How can we compare the simplicity of two design proposals? How do we know what has been left out of the system design?

Experimental research can help to resolve some of these issues and refine our capacity to measure system quality. Still, some aspects of designing will remain an art or intuitive science where esthetics and contemporary style determine success.

The remainder of this paper presents several human factors issues in designing interactive systems. The discussion is independent of hardware-related concerns such as the design of keyboards, displays, cursor controls, audio output, speech recognition, graphics systems, and customized devices, and software-related topics such as natural language front-ends, menu selection, command languages, data-base query facilities, and editors. The emphasis is on general problems and basic experimental results.

First principle: Know the user

Minimize memorization
- Selection not entry
- Names not numbers
- Predictable behavior
- Access to system information

Optimized operations
- Rapid execution of common operations
- Display inertia
- Muscle memory
- Reorganize command parameters

Engineer for errors
- Good error messages
- Engineer out the common errors
- Reversible actions
- Redundancy
- Data structure integrity

Figure 1. User engineering principles for interactive systems (W. J. Hansen, 1971).[7] Hansen's First Principle should be the motto of every designer: Know the User. No qualifier or explanation is necessary. Hansen's sensitivity to human short-term memory limitations leads to his second category: minimizing memorization. Under "optimization of operations," Hansen includes "display inertia," suggesting that when operations are applied, as little of the display should be changed as possible. This approach reduces disruptive movement and highlights the impact of the last operation. "Muscle memory" refers to the idea that users develop the feel for frequently used keypresses. Hansen recognizes the importance of engineering for errors by providing good error messages, reversible actions, and revisions to engineer out common errors.

Attitude and anxiety

Several studies have demonstrated that user attitudes can dramatically affect learning and performance with interactive systems. Walther and O'Neil,[17] for example, showed that novices with negative attitudes towards computers learned editing tasks more slowly and made more errors. Anxiety, generated by fear of failure, may reduce short-term memory capacity and inhibit performance. If users are insecure about their ability to use

1. Provide a program action for every possible type of user input.
2. Minimize the need for the user to learn about the computer system.
3. Provide a large number of explicit diagnostics, along with extensive on-line user assistance.
4. Provide program shortcuts for knowledgeable users.
5. Allow the user to express the same message in more than one way.

Figure 2. The design of idiot-proof interactive programs (A. I. Wasserman, 1973).[8] Wasserman's five design principles are reasonable, but the second and fifth ones may need qualification. Although it is usually good to minimize the user's need to learn about the computer system, restricting access to those who have acquired a certain knowledge level may sometimes be a good idea. The qualifying test, which works well for driver's licensing and college entrance, may be useful for complex and powerful systems. Naive users should be prevented from using a system which is too hard for them and would produce an unpleasant experience. Wasserman's fifth principle may not always be good advice. Novices will prefer and do better with a system which has few choices and permits only limited forms of expression.

the system, worried about destroying files or the computer itself, overwhelmed by volumes of details or pressured to work rapidly, their anxiety will lower performance. Programmers who must meet a deadline tend to make more errors as they frantically patch programs in a manic attempt to finish. Of course, mild pressure can act as a motivator, but if the pressure becomes too strong the resultant high levels of anxiety interfere with competent work.

In designing a system for novices, every attempt should be made to make the user at ease, without being patronizing or too obvious. A message telling users not to be nervous is a bad idea. Users will feel best if the instructions are lucid, in familiar terms, and easy to follow. They should be given simple tasks and gain the confidence that comes with successful use of any tool or machine. Diagnostic messages should be understandable, nonthreatening, and low-key. If the imput is incorrect, avoid blaring phrases such as "ERROR 435—NUMBERS ARE ILLEGAL" and merely state what is necessary to make things right—e.g., "MONTHS ARE ENTERED BY NAME." Try to avoid meaningless, condemning

1. Know the user population.
2. Respond consistently and clearly.
3. Carry forward a representation of the user's knowledge basis.
4. Adapt wordiness to user needs.
5. Provide the users with every opportunity to correct their own errors.
6. Promote the personal worth of the individual user.

Figure 3. Design guidelines for interactive systems (R. W. Pew and A. M. Rollins, 1975).[9] Pew and Rollins echo Hansen's motto and add some of their own besides. Guideline No. 4, above, was probably intended to mean "adapt the messages to the user's level of syntactic and semantic knowledge."

Introduce through experience

Immediate feedback

Use the user's model

Consistency and uniformity

Avoid acausality

Query-in-depth (tutorial aids)

Sequential—parallel tradeoff
(allow choice of entry patterns)

Observability and controllability

Figure 4. Guidelines for designing interactive systems (Brian R. Gaines and Peter V. Facey, 1975).[10] Gaines and Facey emphasize the importance of the user being in control of the terminal, the pace of the interaction, the tutorial aids, and the execution process.

Simple: project a "natural," uncomplicated "virtual" image of the system.

Responsive: respond quickly and meaningfully to user commands.

User-controlled: all actions are initiated and controlled by the user.

Flexible: flexibility in command structures and tolerance of errors.

Stable: able to detect user difficulties and assist him in returning to correct dialogue; never "dead ending" the user (i.e., offering no recourse).

Protective: protect the user from costly mistakes or accidents (e.g., overwriting a file).

Self-documenting: the commands and system responses are self-explanatory and documentation, explanations, or tutorial material are part of the environment.

Reliable: not conducive to undetected errors in man-computer communication.

User-modifiable: sophisticated users are able to personalize their environment.

Figure 5. Interface design for time-sharing systems (D. R. Cheriton, 1976).[11] Cheriton's thorough list provides good guidelines for interactive system designers.

- Simplicity
  - Few keywords
  - Simplicity of input
  - Short commands
  - Simple commands
- Clarity
  - Hierarchical structure (commands and subcommands)
  - Functional separation of commands
  - Homogeneity (same structure for all commands)
  - Problem orientation
- Uniqueness
  - Determinism—every command is fully determined by its operands and preset options
  - No undefined states
- Comfortable language
  - Powerful commands
  - Flexibility
  - Short dialogue
  - Data structures can be displayed and utilized for searching and browsing
- Other comfort
  - Input comfort: rereading or previous input or output after corrections have been made; menu technique
  - Dialogue can be interrupted at any time
  - Clear, short, understandable system messages
- Evidence and reusability
  - Evidence of the system state
  - Acknowledgment of executed commands
  - Help functions
  - Former commands and output reusable for input
  - Saving commands for later execution
- Stability
  - Clear messages on severe input errors
  - Error correction on slight errors
  - Uniform error handling
  - No compulsion to continue the dialogue in a fixed way
- Data security

Figure 6. Design criteria for documentation retrieval languages (F. Gebhardt and I. Stellmacher, 1978).[12]

Forgiveness—ease in repairing errors

Segmentation—layered approach

Variety—choice of style

Escape—break out of danger

Guidance—direction and learning

Leverage—flexible, powerful features

Figure 7. Human/machine interface design criteria in a computerized conferencing environment (M. W. Turoff, J. L. Whitescarver, and S. R. Hiltz, 1978).[13]

Use terse "natural" language, avoid codes, allow abbreviations.

Use short entries to facilitate error correction and maintain tempo.

Allow user choice of single or multiple entries.

Maintain "social element" to the communication.

Permit user to control length of cues or error messages.

Error messages should be polite, meaningful, and informative.

Give help when requested or when users are in difficulty.

Simple, logically consistent command language.

Control over all aspects of the system must appear to belong to the user.

Avoid redundancy in dialogue.

Adapt to the user's ability.

Keep exchange rate in user's stress-free range; user can control rate.

Figure 8. Ground rules for a "well-behaved" system (T. C. S. Kennedy, 1974).[14] This list is based on experimental studies with data entry.

messages such as "SYNTAX ERROR" and give helpful, informative statements such as "UNMATCHED RIGHT PARENTHESIS." Constructive messages and positive reinforcement produce faster learning and increase user acceptance.

Control

A driving force in human behavior is the desire to control. Some individuals have powerful needs to attain and maintain control of their total environment; others are less strongly motivated in this direction and are more accepting of their fate. With respect to using computers, the desire for control apparently increases with experience. Novice terminal users and children are perfectly willing to follow the computer's instructions and accept the computer as the controlling agent in the interaction. With experience and maturity, users resent the computer's dominance and prefer to use the computer as a tool. These users perceive the computer as merely an aid in accomplishing their own job or personal objectives and resent messages which suggest that the computer is in charge.

The Library of Congress recognized this distinction in changing the prompting message from the authoritarian "ENTER NEXT COMMAND" to the servile "READY FOR NEXT COMMAND." A large bank offers a banking terminal which displays the message "HOW CAN I HELP YOU?" This is appealing at first glance, but after some use, this come-on becomes annoying. The illusion that the machine is just like a human teller is perceived as a deception and the user begins to wonder about other ways in which the bank has been deceptive. The attempt to dominate the interaction, by implying that the terminal will help the user by emphasizing the "I," violates common rules of courtesy. If a starting message is used at all, it probably should focus on the customer—for example, "WHAT DO YOU NEED?" followed by a list of available operations. In any case the user should initiate the operation by hitting a button labeled "START," thus reinforcing the idea that the user is in control of the machine.

Early computer-assisted instruction systems heaped praise on the student and "wisely" guided the student through the material at a computer-selected pace; more recent systems merely display performance scores and provide an environment where the student chooses the path and pace. Only children appreciate praise from a computer; most people achieve internal satisfaction if their performance is satisfactory. Instead of the lengthy "VERY GOOD, YOU GOT THE RIGHT ANSWER," the simple display of "++" signals a correct answer to a problem.

Reinforcement for these ideas comes from Jerome Ginsburg of the Equitable Life Assurance Society, who prepared an in-house set of guidelines for developing interactive applications systems. He makes the powerful claim that

> Nothing can contribute more to satisfactory system performance than the conviction on the part of the terminal operators that they are in control of the system and not the system in control of them. Equally, nothing can be more damaging to satisfactory system operation, regardless of how well all other aspects of the implementation have been handled, than the operator's conviction that the terminal and thus the system are in control, have "a mind of their own," or are tugging against rather than observing the operator's wishes.

Being in control is one of the satisfying components of time-sharing and of programming in general. Systems which are designed to enhance user control are preferred. One explanation of why word processing systems have come into widespread use in only the last few years is that mini and microcomputers give users a powerful feeling of being in control compared to the time-shared usage of a large machine. Files kept on floppy disks are tangible when compared to disk files on an unseen remote machine. Although failures, loss of files, and faulty disks probably occur more often on the stand-alone minis and

micros than on larger systems, the users of minis and micros have the satisfaction of controlling their own destiny.

Closure

One of the byproducts of the limitation on human short-term memory is that there is great relief when information no longer needs to be retained. This produces a powerful desire to complete a task, reduce our memory load, and gain relief. *Closure* is the completion of a task leading to relief. Since terminal users strive for closure in their work, interactions should be defined in sections so completion can be attained and information released. Every time a user completes editing a line or ends an editing session with an EXIT or SAVE command, there is relief associated with completion and attaining closure.

The pressure for closure means that users, especially novices, may prefer multiple small operations to a single large operation. Not only can they monitor progress and ensure that all is going well, but they can release the details of coping with early portions of the task. One informal study showed that users preferred three separate menu lists rather than three menus on the screen at once. Although more typing and more interactions were required for the three separate menus, the users preferred doing one small thing at a time. With three menus at a time, the information about the first menu decision must be maintained until the system acknowledges or the RETURN key is hit. Similarly, word processor users may make three separate changes on adjacent words, when one large change command could have accomplished the same results with fewer keystrokes.

Response time

Most designers recognize that a simple limit on response time, the time it takes for the system to respond to a command (e.g., two seconds), is an unreasonably crude specification. Some systems have design specifications of two-second response time for 90 percent of the commands and 10-second response time for the remaining 10 percent. A more informed view is that the acceptable response time is a function of the command type. Users are not disturbed to wait several seconds for the loading of a file or large program, but expect immediate response to editing commands or emergency requests. R. B. Miller[18] provides a list of 17 command types and reasonable response times (Table 1). We may disagree with specific entries or suggest new entries, but the idea of having different response times seems acceptable. In fact, one possible approach is to guarantee that more complex and expensive commands require longer waits. This will tend to make users favor faster, cheaper commands.

A contrasting design goal is to minimize the variance of response time. It has been confirmed by experiment[19] that increasing the variability of response time generates poorer performance (Figure 9) and lower user satisfaction (Figure 10). Users may prefer a system which always responds in 4.0 seconds to one which varies from 1.0 to 6.0 seconds, even through the average in the second case is 3.5. Apparently users can devote 3.9 seconds to planning if they are sure that the time is available. If attention has to be maintained on the screen, users will not use the response time for planning work. Some users even report surprise and disruption if the response is too prompt. Holding responses to minimize response time variance may actually improve user performance and satisfaction. For extremely long response times—i.e., more than 15 seconds—the user should be informed of the time required. One graphics system shows a clock hand ticking backwards counting off the seconds until the system will respond. Even if the response is ready earlier, the system continues its countdown to zero.

Table 1.
System response times as function of user activity (R. B. Miller, 1968).[18]

USER ACTIVITY	"MAXIMUM" RESPONSE TIME (SECONDS)
CONTROL ACTIVATION (FOR EXAMPLE, KEYBOARD ENTRY)	0.1
SYSTEM ACTIVATION (SYSTEM INITIALIZATION)	3.0
REQUEST FOR GIVEN SERVICE:	
SIMPLE	2.0
COMPLEX	5.0
LOADING AND RESTART	15.0-60.0
ERROR FEEDBACK (FOLLOWING COMPLETION OF INPUT)	2.0-4.0
RESPONSE TO ID	2.0
INFORMATION ON NEXT PROCEDURE	< 5.0
RESPONSE TO SIMPLE INQUIRY FROM LIST	2.0
RESPONSE TO SIMPLE *STATUS* INQUIRY	2.0
RESPONSE TO COMPLEX INQUIRY IN TABLE FORM	2.0-4.0
REQUEST FOR NEXT PAGE	0.5-1.0
RESPONSE TO "EXECUTE PROBLEM"	< 15.0
LIGHT PEN ENTRIES	1.0
DRAWINGS WITH LIGHT PENS	0.1
RESPONSE TO COMPLEX INQUIRY IN *GRAPHIC* FORM	2.0-10.0
RESPONSE TO DYNAMIC MODELING	—
RESPONSE TO GRAPHIC MANIPULATION	2.0
RESPONSE TO USER INTERVENTION IN AUTOMATIC PROCESS	4.0

Installers of time-sharing systems report user dissatisfaction in two situations where response time variance is a factor. In the first case, when a new time-sharing system is installed and the workload is light, response times are low and users are pleased. As the load increases, the response time will deteriorate to normal levels and produce dissatisfaction. By slowing down the system when it is first installed, the change is eliminated and users seem content. A second case occurs when the load on a time-sharing system varies substantially during the day. Users become aware of the fast and slow periods and try to cram their work into the fast periods. Although this approach does help to balance the load, users tend to make errors while working quickly to beat the crowd. Anxiety is increased, complaints increase, and programmers or terminal users may even be unwilling to work during the slow periods. By eliminating the variance in response time, service is perceived to be more reliable and one source of anxiety can be reduced.

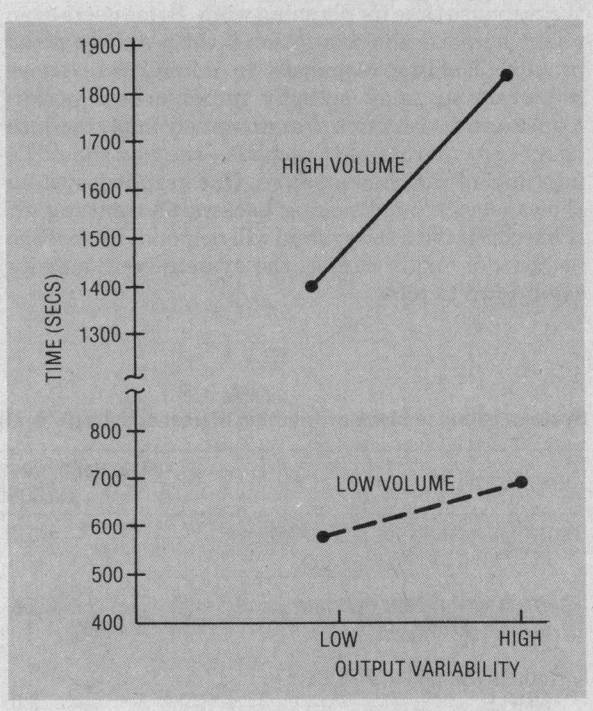


Figure 9. Graph of time to complete tasks vs. output variability for low volume and high volume. (L. H. Miller, 1977).[19]

In summary, response time is an intriguing issue whose complexities have not yet been unraveled. We are left with several conflicting design goals:

- Response time should be reduced under all conditions.
- Response time should match the complexity and cost of the command.
- Variance of response time should be reduced even at the expense of some increase in mean response time.
- System performance should not vary over time.

In an experiment studying the effect of system response time on performance in a multi-parameter optimization task, solution time increased significantly with system response time.[20] Subjects modified five parameters with light pen touches till a curve matched requirements. Each of the 30 subjects performed the task with fixed system response times of 0.16, 0.72, and 1.49 seconds. Figure 11 shows that decreasing the response time from 1.49 to 0.72 seconds reduces the solution time for this task.

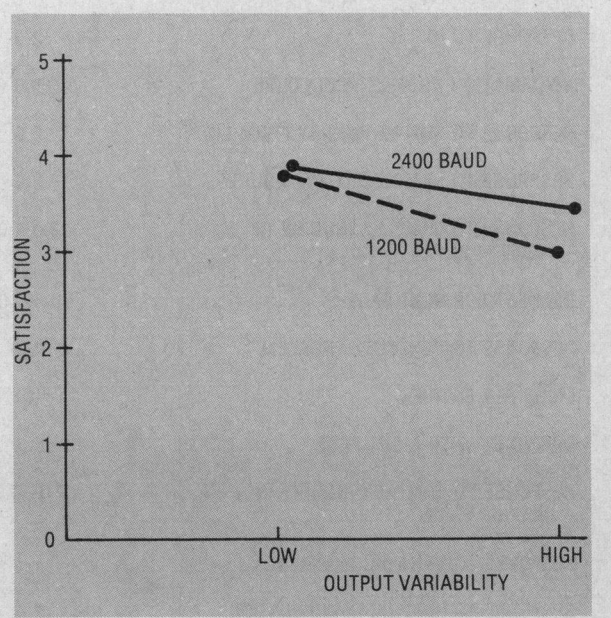


Figure 10. Graph of average response to post-test questionnaire vs. output variability for 1200 and 2400 baud (L. H. Miller, 1977).[19]

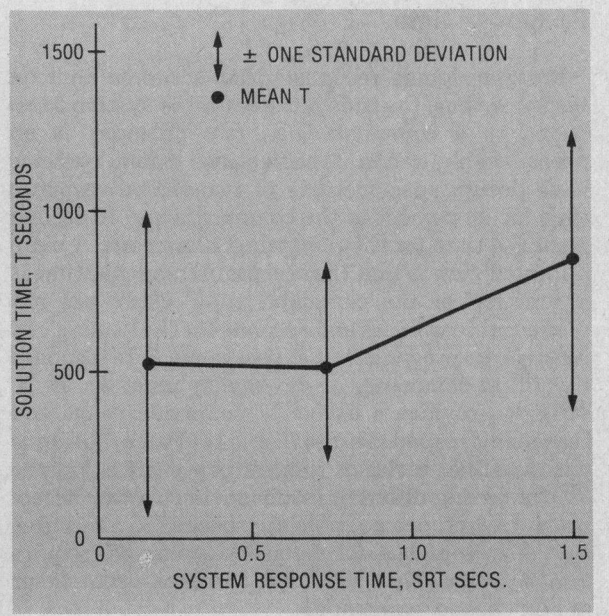


Figure 11. Solution time (T) versus System Response Time (SRT) for 30 subjects (Goodman and Spence, 1978).[20]

Grossberg, Wiesen, and Yntema[21] studied four subjects performing 36 interactive tasks involving calculations on numeric arrays. Response times were varied from 1 to 4 to 16 to 64 seconds. As the response time increased subjects became more cautious, used fewer commands, and took longer time between commands, but the total time consumed showed surprising invariance with respect to the response time increase. The subjects changed their working style as the response time increased by becoming more cautious and by making heavier use of hard copy printouts. The difference in results between this experiment and the previous one may be a product of the available commands, required tasks, or subject experience.

A related aspect of response time is the thought time of the terminal user. For complex decision-making, there is some evidence that locking the terminal for a short period, say 25 seconds in one pilot study, may improve user performance on the decision and increase user satisfaction. An open keyboard and partial attention to the display can distract the users and interfere with problem-solving while increasing anxiety. The illusion of "dialog" may compel users to keep their end of the "conversation" going. A decision-making study[22] with longer lockout times (5 and 8 minutes) revealed that subjects with no lockout used twice as much computer time and, as might be expected, the lockout groups expressed dissatisfaction with restricted access. The high variance in performance of the 20 subjects made it impossible to assess the impact of lockout, although the highest performance mean was achieved by the 5-minute lockout group. Possibly if users perceive the computer as a tool, they may be more willing to take their time and reflect on decisions. If users feel they are involved in a "dialog" in which they must respond promptly, anxiety and poorer performance may result. Maybe we should replace the term "dialog" with "utilog" conveying the impression that the user is utilizing the system as a tool.

Time-sharing vs. batch processing

As technological developments allowed programmers to use interactive terminals for preparing and executing their programs, a controversy arose over the relative merits of interactive usage and traditional batch submittal. Adherents of time-sharing argued that waiting for processing by batch-oriented computer systems was annoying, disruptive, and time-consuming. Others felt that time-sharing encouraged sloppy and hasty programming, which in turn led to more errors and poorer quality work.

Two of the earliest studies comparing on-line and off-line processing were by Schatzoff, Tsao, and Wiig[23] and Gold.[23] The former study showed a 50 percent higher total cost for time-sharing, and a 50-percent greater elapsed time for batch, with no difference in computer time. More compilations were made on-line, suggesting less time is spent in desk checking. According to Gold,[24] the "user's attitude appears to be one of the variables which may influence the user's immediate behavior and usage of computer systems." Both studies agreed that some performance variations may be attributable to programmer and problem differences.

Smith[25] examined the effects of conventional batch versus instant batch (less than 5 minutes). With respect to elapsed time (time from the start of a problem to its completion) and student reaction, instant surpassed conventional.

Summarizing five studies comparing on-line to off-line problem solving (including the two mentioned above), Sackman[26,27] stated that time-sharing had a 20-percent advantage over batch in hours used, whereas batch surpassed time-sharing with a 40-percent advantage in CPU time. In regard to cost, neither mode outperformed the other. Sackman suggested that "the comparison ... is becoming academic as the contest converges toward interactive time-sharing and fast or even instant batch." These studies need to be reevaluated and redone since hardware speeds and software capabilities have changed substantially in the last decade.

As a result of experimentation with junior college students, the use of time-sharing was recommended to alleviate the high drop-out rate from the introductory computer science courses.[28] The immediate feedback of time-sharing was seen as positively reinforcing.

The decrease in literature comparing the two modes of program development and the increase in articles on time-sharing systems give the illusion that the controversy has ended and the superiority of on-line processing is accepted. But some managers and researchers suggest that time-sharing mode encourages hasty program development and increases the number of errors. They feel that the slower turnaround of batch processing produces more careful program design and thorough desk debugging.

In a related application of interactive systems, J. V. Hansen[29] investigated performance differences for two management decision-making tasks using time-sharing and batch approaches. Both problems, stochastic capital budgeting and product demand forecasting, were not solvable by a mathematical algorithm. Instead, they required heuristic approaches where feedback from each interaction would suggest new decision rules. The results (Table 2) demonstrate that in this environment time-sharing

Table 2.
Decision-making performance averages using time-sharing and batch modes (J. V. Hansen, 1976).[29]

	GROUP A (BATCH/ON-LINE) (5 SUBJECTS)	GROUP B (ON-LINE/BATCH) (5 SUBJECTS)
PROBLEM 1 (CAPITAL BUDGETING)	82.0 (BATCH)	88.4 (ON-LINE)
PROBLEM 2 (PRODUCT DEMAND FORECAST)	90.6 (ON-LINE)	84.6 (BATCH)

access significantly improved the quality of the decisions.

In short, the experimental results suggest that a good time-sharing system is better than a bad batch system. Correcting minor errors quickly in time-sharing mode speeds productivity and reduces irritation. For more fundamental work, some programmers may abuse the rapid access of time-sharing, make hasty patches, and produce poor code.

In all the experimental results, the influence of individual differences apparently played a major role. The high variance in performance and conflicting anecdotal evidence suggests that unmeasured factors such as personality may influence preference and performance. Whether or not a programmer wants to use interactive equipment may be an important consideration. Merely because many programmers, perhaps even a majority, prefer interactive mode does not mean that all programmers should utilize that mode. Those individuals who feel more secure with a deck of keypunch cards are just as necessary to an organization.

Many variables enter into a programmer's preference for a particular computer communication alternative. In an effort to identify specific personality traits influencing preference, Lee and Shneiderman[30] studied locus of control and assertiveness.

Locus of control focuses on the perception individuals have of their influence over events. Internally controlled individuals perceive an event as contingent upon their own action, whereas externally controlled people perceive a reinforcement for an action as being more a result of luck, chance, or fate; under the control of other powerful people; or unpredictable.

Assertive behavior "allows an individual expression in a manner that fully communicates his personal desires without infringing upon the right of others."[31] Assertive individuals can state their feelings; nonassertive people have difficulty doing so.

Table 3.
Preference scores and personality factors (Lee and Shneiderman, 1978).[30]

	BATCH 0	1	2	3	TIME SHARING 4	MEAN	TOTAL OBSERVATIONS
LOCUS DIMENSION:							
INTERNAL	0	0	2	2	2	3.0	6
EXTERNAL	0	0	8	4	0	2.3	12
							18
ASSERTIVENESS DIMENSION:							
LOW	0	0	5	3	0	2.4	8
HIGH	0	0	5	3	2	2.7	10
							18

Many programmers learned use of keypunch equipment before being introduced to time-sharing. It would be less anxiety provoking for them to remain with a mode of program entry which is familiar—i.e., keypunch—than to attempt on-line communication with its many problems—e.g., signing on or possible loss of an editing session. It seems that individuals who view themselves as more effective and powerful, or internally controlled, would master on-line interaction with the computer, while those who see themselves as less powerful and not very independent or effective, or externally controlled, would continue to process by batch.

Likewise, more assertive programmers would not let the intimidating terminal inhibit them from learning and using interactive equipment. They would be able to ask for help when needed, thus promoting their learning process. The nonassertive individual might look for a means of program entry which allows least contact with others, including avoidance of equipment which could require a great deal of help and guidance during the familiarization stage. Weinberg[32] conjectures that "humble programmers perform better in batch environments and assertive ones will be more likely to shine on-line."

Subjects for our exploratory study were programmers from a Control Data Corporation installation, which allows the choice of either card or terminal entry. Three questionnaires, one to measure locus of control, one to ascertain assertiveness, and another to determine on-line or off-line preference were distributed via interoffice mail.

When the 18 responses were grouped by preference scores (Table 3), the batch group did not differ significantly from the interactive group on either personality dimension: locus of control or assertiveness. However, when the sample was grouped by internal locus/high assertive and external locus/low assertive (Table 4), there was a significant difference in mean preference scores. Confirming studies need to be carried out with more subjects in a wide variety of programming environments.

Although our findings in this exploratory study showed mixed results, the import lies in the attempt to identify variables entering into a programmer's preference for either batch or time-sharing. If programmers are allowed to use the mode they prefer, their performance and job attitude could improve. If

Table 4.
Average preference scores for personality groups (Lee and Shneiderman, 1978).[30]

	INTERNAL LOCUS/ HIGH ASSERTIVE	EXTERNAL LOCUS/ LOW ASSERTIVE
MEAN PREFERENCE SCORE	3.34	2.54
VARIANCE	0.399	0.108
NUMBER OF SUBJECTS (TOTAL NUMBER WAS 18)	4	6

preference is affected by the type of task, the availability of different modes may again improve performance. When recruiting programmers for a time-sharing environment, managers may find that those who desire to work on-line will produce better products in that environment than those who prefer working in a batch environment.

Text editor usage

A rapidly growing mode of computer use is by way of text editors, document preparation systems, and word-processing equipment. These tools allow users to construct files containing programs, alphanumeric data, correspondence, or general textual information. The diversity of user experience and the range of user patterns is enormous. Sophisticated frequent users differ from infrequent users, who are all very different from novice users. The variety of hardware and software environments further increases the choices for text editor designers and users.

Experimental comparisons of text editors are providing information about usage patterns, suggesting directions for development projects, and aiding development of a cognitive model. Walther and O'Neil[17] report on an experiment with 69 undergraduate computer science students: 41 percent had never used an on-line system, 38 percent had some experience, and 22 percent had much experience. The three experimental factors were flexibility (one version of the editor was inflexible; the second version permitted abbreviations, default values, user declaration of synonyms, a variety of delimiters, and other features), display device (cathode ray tube and impact teletype, both at 10 cps), and attitude (three subjective tests indicating attitude towards computers and anxiety). The subjects performed 18 corrections to a text file while errors were tabulated and timing data was collected. Experienced users worked faster with the flexible version, but inexperienced users were overwhelmed by the flexible version. The inexperienced users made fewer errors and worked faster with the inflexible version. The impact teletype users worked faster and made fewer errors, suggesting that the feedback from the impact may facilitate performance. Those with negative attitudes made more errors. Walther and O'Neil offer interaction effects, conjectures, potential design rules, and research directions.

Sondheimer[33] describes an experiment with more than 60 professional programmer users of a text editor. With active participation of the subjects, five features were chosen for addition to the text editor. Announcements, documentation, and training were provided, but after some initial testing, usage of the features dropped off substantially. Sondheimer concludes that "the results of the experiment seem to indicate the persistence of individual usage habits." This experiment has implications which go beyond the use of text editors, but it does emphasize that text editing is a skill which is deeply ingrained in the user's mind and difficult to change. Sondheimer conjectures that novice users of the text editor would more frequently employ the newly added features.

Card[34] and Card, Moran, and Newell[35,36] provide detailed reports on text editor experiments and offer cognitive models of human performance. Their experiments emphasize in-depth study of a limited number of highly trained subjects. Subjects performed manuscript editing tasks with a variety of line and display editors while precise timing measurements were made automatically. Text editing is characterized as a "routine cognitive skill" which "occurs in situations that are familiar and repetitive, and which people master with practice and training, but where the variability in the task, plus the induced variability arising from error, keeps the task from becoming completely routine and requires cognitive involvement."[35] A cognitive model based on goals, operators, methods, and selection rules (GOMS model) is proposed and is claimed to represent the performance of expert users. User style in locating a line (by jumping ahead a given number of lines or by locating a character string) and correcting text (by substitution or by subcommands for modifying characters in a line) was compared among subjects with the goal of predicting behavior in future situations.

Card, Moran, and Newell[35] use data from 28 subjects, on 10 systems, and over 14 task types to support the keystroke model of editor usage, suggesting that task performance time can be predicted from a unit task analysis and the number of keystrokes required. This model has strict requirements: "The user must be an expert; the task must be a routine unit task; the method must be specified in detail; and the performance must be error-free." The timing data from a variety of users and systems reveals important differences, such as the speed advantage of display editors over line editors (about twice as fast). The timing data from Card[33] demonstrates the clear speed and accuracy advantages of a mouse for selecting text, when compared with a joystick, step keys, or text keys.

Error handling

The error-checking and handling components of an on-line system may occupy the majority of the programming effort. Well-designed diagnostic facilities and error messages can make a system appealing. When user entries do not conform to expectations, diagnostic messages should guide the user in entering correct commands. Messages should be brief, without negative tones, and should be constructive. Avoid ringing bells and bold messages which may embarrass the user. Instead of meaningless messages like "ILLEGAL SYNTAX," try to indicate where the error occurred and what may be done to set it right. If possible, allow users to modify the incorrect command rather than forcing complete reentry. Command and programming languages should be designed so that a common error will not be interpreted as a valid command.

Error messages should be included in the system documentation, so that users know what to expect and so that designers cannot hide sloppy work in the system code.

The system should permit easy monitoring of error patterns so that system design can be modified in response to frequent errors. Simple tallies of error occurrences may suggest modifications of error messages, changes to command languages, or improved training procedures.

An intriguing issue in error handling is whether the error message should be issued immediately or when the end-of-line code (usually ENTER or RETURN key) is hit. A nicely designed study[37] suggests that human performance improves if errors are issued immediately and that the disruption of user thought processes by immediate interruption is not a serious impediment. Seventy undergraduate subjects in this experiment had to list 25 of the 50 states in the USA and list 20 permutations of "abcde" such that "c" occurs somewhere before the "d." The results of the permutation task strongly favor immediate interruption, but the results of the states task were mixed (Table 5). A powerful advantage of immediate interruption is that changes can be made simply by replacing the incorrect character.

A central problem in handling errors is providing the user with the right kind of information. Experienced frequent users need only an indication that an error has occurred, such as a locked keyboard, a light, or a special character. As soon as the error has been brought to their attention, they will probably recognize it and be prepared to make an immediate correction. Typical users familiar with the operations or semantics of the domain merely require a brief note to remind them of proper syntax or list of available options. Novice users whose semantic knowledge is shallow need more than prompting on syntax; they need explanations of possible commands and the required syntax. Since even experts may forget or be novices with respect to some portions of a system, a simple scheme based on recording user experience levels is unworkable. Probably the best approach is to give control to the user and provide options—maybe "?" for a brief prompt about syntax, a second "?" for a brief prompt about semantics, and a third "?" for a more detailed explanation. Users could strike "??" or "???" initially to get complete information right away.

This question mark scheme is a simple approach to what are generally referred to as "HELP" systems. Typing "HELP" or merely "H" the user can get some information; "HELP FILES," "HELP EDIT," "HELP FORTRAN," etc., may invoke more extensive topic-oriented HELP facilities. "HELP HELP" should provide information about available facilities. The PLATO instructional system offers a special HELP key which offers appropriate guidance for the material currently on the screen.

Table 5.
Average performance results to error correction styles (Segal, 1975).[37]

	STATES TASK		PERMUTATION TASK	
	ERROR CORRECTION METHOD			
	IMMEDIATE	END	IMMEDIATE	END
PERCENT ERROR KEYPRESSES	2.55	1.99	4.54	4.48
TOTAL TIME (SECONDS)	234.0	300.0	408.8	546.4
TWO CONSECUTIVE RESPONSES IN ERROR	1.17	1.17	1.00	2.77
NUMBER OF RESPONSES IN ERROR	4.29	3.77	4.46	4.83

Practitioner's summary

Do not violate the bounds of human performance imposed by limited short-term memory capacity. Design interactions in a modular fashion so that closure can be obtained providing satisfaction and relief for users. Be sensitive to user anxiety and desire for control. Provide novice users with the satisfaction of accomplishment and a sense of mastery, but avoid patronizing comments. Consider response time requirements as part of the design, not as an uncontrollable aspect of system performance.

Respect user preferences in choice of batch or interactive program development. Accept the personality and cognitive style differences among individuals and do not attempt to make everyone behave as you do.

Devote substantial energy to error design. Make messages constructive and give guidance for using the system in a courteous nonthreatening way. Prepare all messages as part of the system design and make them available in user manuals. Give users control over what kind of and how much information they wish at every point in the interaction. Do not require them to identify themselves at the start as novices. HELP facilities should be available for every command.

Respect and nurture the user community. Listen to their gripes with sympathy and be willing to modify your system to accommodate their requests. Remember, the goal is not to create a computerized system, but to serve the user. ■

Acknowledgments

This article was adapted from portions of the forthcoming book, *Software Psychology: Human Factors in Computer and Information Systems* (Winthrop Publishers). The support for computer resources was provided by the University of Maryland Computer Science Center. Helpful comments on earlier drafts of this paper were provided by Jim Foley, John Gannon, Tom Love, G. Michael Schneider, John C. Thomas, Jr., and Jerry Weinberg.

References

1. Henry Dreyfuss, *Designing for People,* Simon and Schuster, New York, 1955.

2. R. S. Nickerson, "Man-Computer Interaction: A Challenge for Human Factors Research," *IEEE Trans. Man-Machine Studies,* MMS-10, 1969, p. 164.
3. J. L. Bennett, "The User Interface in Interactive Systems," in C. Cuadra (ed.), *Annual Review of Information Science and Technology,* Vol. 7, American Society for Information Science, Washington, D.C., 1972, pp. 159-196.
4. J. Martin, *Design of Man-Computer Dialogues,* Prentice-Hall, Englewood Cliffs, N. J., 1973.
5. L. A. Miller and J. C. Thomas, Jr., "Behavioral Issues in the Use of Interactive Systems," *Int'l J. Man-Machine Studies,* Vol. 9, 1977, pp. 509-536.
6. B. Shneiderman, "Improving the Human Factor Aspect of Database Interactions," *ACM Trans. on Database Systems,* Vol. 3, No. 4, Dec. 1978, pp. 417-439.
7. W. J. Hansen, "User Engineering Principles for Interactive Systems," *AFIPS Conf. Proc.,* Vol. 39, 1971 FJCC, AFIPS Press, Montvale, N. J., 1971, pp. 523-532.
8. A. I. Wasserman, "The Design of Idiot-Proof Interactive Systems," *AFIPS Conf. Proc.,* Vol. 42, 1973 NCC, AFIPS Press, Montvale, N. J., 1973, pp. M34-M38.
9. R. W. Pew and A. M. Rollins, "Dialog Specification Procedure," Bolt Beranek and Newman, Report No. 3129, Revised Ed., Cambridge, Mass., 1975.
10. Brian R. Gaines and Peter V. Facey, "Some Experience in Interactive System Development and Application," *Proc. IEEE,* Vol. 63, No. 6, June 1975, pp. 894-911.
11. D. R. Cheriton, "Man-Machine Interface Design for Time-Sharing Systems," *Proc. ACM Nat'l Conf.,* 1976, pp. 362-380.
12. F. Gebhardt and I. Stellmacher, "Design Criteria for Documentation Retrieval Languages," *J. Am. Soc. Information Science,* Vol. 29, No. 4, July 1978, pp. 191-199.
13. M. W. Turoff, J. L. Whitescarver, and S. R. Hiltz, "The Human Machine Interface in a Computerized Conferencing Environment," *Proc. IEEE Conf. on Interactive Systems, Man, and Cybernetics,* 1978, pp. 145-157.
14. T. C. S. Kennedy, "The Design of Interactive Procedures for Man-Machine Communication," *Int'l J. Man-Machine Studies,* Vol. 6, 1974, pp. 309-334.
15. J. D. Foley and V. L. Wallace, "The Art of Graphic Man-Machine Conversation," *Proc. IEEE,* Vol. 62, No. 4, Apr. 1974, pp. 462-471.
16. Stephen E. Engel and Richard E. Granda, "Guidelines for Man/Display Interfaces," IBM Poughkeepsie Laboratory Technical Report TR 00.2720, Dec. 19, 1975.
17. G. H. Walther and H. F. O'Neil, Jr., "On-line User-Computer Interface—the Effect of Interface Flexibility, Terminal Type, and Experience on Performance," *AFIPS Conf. Proc.,* Vol. 43, 1974 NCC, pp. 379-384.
18. Robert B. Miller, "Response Time in Man-Computer Conversational Transactions," *AFIPS Conf. Proc.,* Vol. 33, 1968 SJCC, pp. 267-277.
19. L. H. Miller, "A Study in Man-Machine Interaction," *AFIPS Conf. Proc.,* Vol. 46, 1977 NCC, pp. 409-421.
20. T. Goodman and R. Spence, "The Effect of System Response Time on Interactive Computer-Aided Problem Solving," *Proc. ACM SIGGRAPH '78,* pp. 100-104.
21. Mitchell Grossberg, Raymond A. Wiesen, and Douwe B. Yntema, "An Experiment on Problem Solving with Delayed Computer Responses," *IEEE Trans. Systems, Man, and Cybernetics,* Vol. SMC-6, No. 3, Mar. 1976, pp. 219-222.
22. B. W. Boehm, M. J. Seven, and R. A. Watson, "Interactive Problem-Solving—An Experimental Study of 'Lockout' Effects," *"AFIPS Conf. Proc.,* Vol. 36, 1971 SJCC, pp. 205-210.
23. M. Schatzoff, R. Tsao, and R. Wiig, "An Experimental Comparison of Time-Sharing and Batch Processing," *Comm. ACM,* Vol. 10, No. 5, May 1967, pp. 261-265.
24. M. M. Gold, "Time-Sharing and Batch Processing: An Experimental Comparison of their Value in a Problem-Solving Situation," *Comm. ACM,* Vol. 12, No. 5, May 1969, pp. 249-259.
25. L. B. Smith, "A Comparison of Batch Processing and Instant Turnaround," *Comm. ACM,* Vol. 10, No. 8, Aug. 1967, pp. 495-500.
26. H. Sackman, "Experimental Analysis of Man-Computer Problem-Solving," *Human Factors,* Vol. 12, 1970, pp. 187-201.
27. H. Sackman, *Man-Computer Problem Solving,* Auerbach Publishers Inc., Princeton, N. J., 1970.
28. Michel Boillot, "Computer Communication Modes and their Effect on Student Attitudes Towards Programming," Nova University thesis, available through ERIC ED 098 957, Apr. 1974.
29. J. V. Hansen, "Man-Machine Communication: An Experimental Analysis of Heuristic Problem-Solving under On-Line and Batch-Processing Conditions," *IEEE Trans. Systems, Man and Cybernetics,* Vol. 6, No. 11, Nov. 1976, pp. 746-752.
30. Jeanne M. Lee and B. Shneiderman, "Personality and Programming: Time-Sharing vs. Batch Processing," *Proc. ACM Nat'l Conf.,* 1978, pp. 561-569.
31. B. J. Winship and J. D. Kelly, "A Verbal Response Model of Assertiveness," *Counseling Psychology,* Vol. 23, No. 3, 1976, pp. 215-220.
32. G. M. Weinberg, *The Psychology of Computer Programming,* Van Nostrand Reinhold, N. Y., 1971.
33. Norman Sondheimer, "On the Fate of Software Enhancements, *AFIPS Conf. Proc.,* Vol. 48, 1979 NCC, pp. 1043-1048.
34. Stuart K. Card, "Studies in the Psychology of Computer Text Editing," Xerox Palo Alto Research Center, SSL-78-1, San Jose, Calif., Aug. 1978.
35. Stuart K. Card, Thomas P. Moran, and Allen Newell, "The Keystroke-Level Model of User Performance Time with Interactive Systems" (submitted for publication).
36. Stuart K. Card, Thomas P. Moran, and Alan Newell, "Computer Text-Editing: An Information-Processing Analysis of a Routine Cognitive Skill," *Cognitive Psychology* (to appear).
37. Barr Zion Segal, "Effects of Method of Error Interruption on Student Performance at Interactive Terminals," University of Illinois Department of Computer Science Technical Report UIUCDCS-R-75-727, May 1975.

Ben Shneiderman, an associate professor at the University of Maryland, has produced five books (including the forthcoming *Software Psychology)* plus 50 articles on data-base management, programming, and human factors. After receiving his PhD in computer science from the State University of New York at Stony Brook, he spent three years teaching in the Department of Computer Science at Indiana University, before coming to the University of Maryland in 1976.