THE
UNIVERSITY
OF CALGARY

Designing for Human-Computer Interaction:

some rules and their derivation.

by

David R. Hill

Research Report 84/166/24

DEPARTMENT OF
COMPUTER SCIENCE

MAN-MACHINE SYSTEMS LABORATORY
Department of Computer Science
The University of Calgary
CALGARY, Alberta, Canada T2N 1N4

Designing for Human-Computer Interaction:

some rules and their derivation.

by

David R. Hill

Research Report 84/166/24

The design rules set out in this document first appeared in this form in:

HILL, D.R., WITTEN, I.H. NEAL, R. & LOMOW, G. (1984) Jecl and Hide: practical questions for the Jade user interface. *Proc. CIPS Session 84, Calgary, May 9-", pp 373-380,* and are based on lecture notes from the computer science course taught by the first author: "**CPSC 481: Human-Computer Interaction**"

The text is a draft of a chapter in a book by the first author in preparation:
Human Computer Interaction: concepts, methods and problems

SUMMARY

There has been an explosion of interest in the human computer interface. For a variety of reasons, mainly economic, companies are searching for better ways for people to interact with information processing systems. One problem has been a lack of awareness amonongst those most likely to benefit from careful interface design, and innovative interfaces, of the real issues and benefits. Even computer experts can reap rewards from improvements in the human computer interface ("programming environments"). Programmers are computer users as well. After considering the design process, in the context of human computer interaction, the paper goes on to spell out major principles for the Interface designers, providing a detailed statement of what is involved in applying the principles, and how these arise on the basis of our knowledge of people and machines. The last section provides an informal discussion of some of the issues and steps involved in designing human-computer interfaces.

Chapter 6: Dialog design

Introduction

In the last year or two. there has been an upsurge of interest in providing better ways for people to interact with information processing systems. There are several reasons for this. First. it has become apparent that poor interfaces make it more difficult for users of computer systems (including computer science experts) to do their job. Better interfaces improve productivity, reduce errors, and allow higher quality results. They also give a competitive edge to their suppliers and make the users more comfortable in their work.

Secondly, with falling hardware costs and rising labour costs, the emphasis has changed from utilising machines to their maximum capacity to utilising their human users and operators to maximum capacity.

Thirdly, computers are becoming very wideiy used, even in areas and in equipment that have previously not been associated with computers. The users of computers, in these circumstances, frequently have little or no computer training, and may exhibit the whole gamut of educational and career achievement in their own specialities. For such people. the computer should appear as a tool, interfaced in such a way that the user can think about the task goals for which the system is used, rather than the characteristics of the computer tool used to achieve these goals. Some systems must carry the computer power so deeply embedded that it is effectively hidden, just as the electric motor in a dishwasher or clock is hidden. The interface seen by the user is completely task-oriented, and the internals of the system translate the user's needs into the control and power signals required to employ the motor as a subsystem. Of course, the user may well be aware that a computer (or motor) is in there doing essential things, but does not have to be concerned with its characteristics.

Thus, so-called *User Friendly Interfaces* have become the touchstone for the more widespread and effective use of computer power. Such interfaces have a direct economic and social impact, to the extent they succeed or fail. They allow the computer industry generally to expand markets, hence creating new jobs within the computer industry. They also allow the market sectors which start applying computer power to be more productive and competitive, which may both expand existing market shares and create new markets in application areas. Jobs may actually be lost in the original applications areas themselves—a clear warning to those responsible for government that they must get into the computer and computer system supply side to maintain a full economy. Those who remain only as end (application) users will not need so many people to do the work, and will also have to spend a great deal of foreign exchange to obtain equipment needed to remain competitive in their traditional areas of operation. Such countries will become increasingly dependent on natural resources (if they have them), and increasingly poor compared to countries that ride the wave, especially If the bulk of the natural resources are non-renewable.

These facts may seem a far cry from computer science, but in reality they are the underlying cause of the explosive (and ultimately unsatisfiable) need for trained computer science specialists to bridge the gap in interface design excellence. Six years ago, the graphics area in computer science exploded dramatically, as the need, the methodology, and the technology appeared or were generated. Advertising, film-making, and design have provided much of the finance and incentive to the graphics explosion. Now that costs have fallen, as research has been amortised. and mass-production has started to appear, graphics is providing part of the base for better human-computer interface design. Other technologies are starting to mature—expert systems, low-cost very powerful

desktop computers with high-resolution colour displays, dialogue prototyping methods, databases and database access, new inexpensive input-output devices, and so on. It is now commonplace to do things that were not possible even as recently as two years ago. Partly this allows new approaches to human computer interfacing, and partly it allows better and more diverse experiments to be performed as part of the research to expand the body of knowledge concerning the methods and goals of human-computer interface practice.

The Apple *Lisa* (Williams 1983. Morgan et et. 1983) is an example of the application of both new technology and new knowledge. Whether it is an experiment or a new tool. only time and the marketplace will tell. However. it is a pretty hefty straw In the wind. and shows just how strongly the wind must be blowing. In the Usa project. it is instructive to note that the entire design team spent the first 6 months designing the *User Manual*, whilst avoiding any actual implementation. This is how the design of such systems must proceed. The Macintosh is a down market popularisation of the techniques and software developed for *Lisa*, and has proved very popular, living up to its claims of ease of use.

One problem that professionals working in the new area of human computer interface design may encounter is that of ignorance on the part of higher management and users. The ignorance referred to is not that these groups lack the specialist knowledge of the professional so much as the fact that they are unaware of the problems caused by failure to deal adequately with the human computer interface, and are unaware of the benefits (especially economic benefits) of proper design. Thus an important part of a professional's job will be education (e.g. McLaughlin, 1985). A number of points arise from this. At the simplest level, users must be made to realise that some of the problems they encounter are design problems rather than their own problems, just as pilots, accused of pilot error, were shown to be incapable of performing some of the operations required by aircraft designers. Users must be educated to criticise poor designs. Programming teams must be educated to accept the idea that designing the system is largely a matter of designing the user interface, and that this activity must precede coding (see below). Perhaps the most difficult problem arises in the case of competitive tenders for systems. There are many examples of systems that were bought, and then never used, or used with disastrous results, due to poor user interface design (e.g. BCS 1976). The problem is that, in a competitive bidding situation, if the Request For Quotation (RFQ) does not include a specification of the user interface, what performance criteria it must meet, and how it will be evaluated and accepted, any reputable company is liable to lose in the bidding because less reputable companies will not include the cost of interface design and testing. Those submitting RFQ's must be educated to include such components in their requests. Furthermore, reputable companies must be prepared to get back to clients submitting RFQ's that omit such material with a refusal to tender in the absence of a revised RFQ, with a careful explanation of the consequences on intractability on this issue. Given the potential legal ramifications of failure of poorly designed systems, this is at least enlightened self-interest.

A first look at the problem

The programmer as a user

Experience has shown that poor interfaces make it more difficult for computer users to do their job. Even computer experts show increased productivity, reduced errors, and higher quality work when they are provided with a better programming

environment and more powerful tools that are easy to apply to their work. Furthermore. falling hardware costs and rising labour costs are shifting the emphasis from machine utilisation to human productivity, in terms of increased throughput, reduced errors, shorter training periods, and lower staff turnover, whilst still maintaining or preferably improving the quality of work produced. With the increasingly widespread use of computers by non-experts, for a variety of economic and practical reasons, these facts have already led to a dramatic surge in the attention given to the human-computer interface in applications areas, especially when advertising products. However. the corresponding rise in our knowledge of how to design good interfaces, even for well defined applications tasks, has been far less than dramatic. For programming there is almost nothing, despite the fact that, in a very real sense, one of the most important applications of computers is to programming. There have been studies and experiments concerned with various aspects of the psychology of programming and much written about the value of structured program design and the relative merits of various kinds and levels of languages. Unfortunately, with honourable exceptions (e.g. Davis 1984). too many writers seem to think that one language, usually their pet, should be used for everything. Sheil (1981) presents an interesting view:

> Most innovations in programming languages and methodology are motivated by a belief that they will improve the performance of the programmers who use them. Although such claims are usually advanced informally, there is a growing body of research which attempts to verify them by controlled observation of programmers' behaviour. Surprisingly, these studies have found few clear effects of changes in either programming notation or practice. Less surprisingly, the computing community has paid relatively little attention to these results.

He goes on in the paper to suggest that the problem is due to the unsophisticated experimental techniques used, and a *shallow view of the nature of programming skill* (our italics).

A fortiori, no systematic study seems to have been made of the overall needs of the programmer as a user of computers. It seems to be assumed that the programmer is so expert that he or she: (a) can take care of him or herself; and (b) has needs so arcane that they are beyond study. The "programmer's operating system", *Unix*, is a fruit of, or caution against this approach, depending on your viewpoint. The original version was written by a programmer, working at Bell Laboratories, strictly for himself. because he had been given a minicomputer with an operating system that did not meet his needs. Subsequently, other programmers liked it so much that it spread, and was then adopted by AT&T, then licensed to universities (with some very expensive commercial licensing), and finally (quite recently) turned into a "product". In many ways *Unix* represents a distillation of what serious computer scientists (who are all programmers ?) need. At least it is the best available. At the same time, it has many of the shortcomings of other applications devised by programmers for users, as Norman has pointed out (Norman in press). Programmers are not necessarily expert in the techniques and pitfalls of good interface design, and (as has been confirmed by our experiencel they may not even have a very clear idea of what they do, let alone the "best" way to do it. In this, they are typical users, albeit with a great deal of computer oriented expertise, including experience of coping with existing facilities. It is partly a lack of understanding of the programmer's task that leads Computer Science Research funding agencies and university administrations to veto the idea of providing certain kinds of facilities (e.g. laser printers and document preparation tools). It is possible that the only important thing that both systems and applications programmers do is to document. It is just that some of the documentation (on how to solve a particular problem) can be directly interpreted and executed by a computer.

However. such ‘documentation’ is very demanding, and requires many special aids, techniques and system facilities to make it easy to produce good examples efficiently. Programmers also have to document at several other levels so that other people (usually also programmers, and therefore assumed infinitely clever by far too many people) can: (a) understand (and therefore check. debug. or modify) the problem solution statement; and (b) implement it in such a way that other people (usually non-programmers) can use the problem solution itself in their work. This latter requires an operating manual, often called a user manual. Most programmers seem to hate documentation. except the kind fed into the computer. so one obvious component of a programming system is something to make program documentation easy. We have yet to see such a creation that works.

In all this. the important case of the *programmer* as a user comes off worst of all. Worse still, too many programmers, who seem to associate the term ‘User Friendly’ with patronising, anthropomorphic, or inefficient interaction, are quite content with—indeed prefer—this outcome. They certainly distinguish themselves sharply from mere “users”. even whilst claiming to know (without much investigation) what a user needs. With increasingly sophisticated tools devoted to allow “end users” (the correct term for the “users” above) to develop their own applications, as well as increasingly knowledgeable end users. the attitude is at best self-defeating.

Two of the earliest papers on interactive system design (Foley & Wallace 1974; Hansen 1971) are still amongst the best introductions to the problems and principles. More recent papers have tended to be a re-statement of the principles, or accounts of particular systems. Norman (1983) has produced an excellent restatement of the issues while MacGuire (1982) provides a concise survey of earlier views. Much can be learned from such accounts, but there is no established design procedure for the human-computer interface, although certain important necessary steps are becoming clear. In the next section we take a broad look at the problem of user interface design, and go on to consider the design process. The follows a section in which we elaborate on the design principles for human-computer interaction, and suggest a relationship between these and the detailed rules needed to instantiate a design. The principles also represent categories of design activity within which to address particular human needs, strengths, or failings. These latter are also spelled out, linked to the principles and practices to which they give rise. It should be remembered that the term “user” is intended to include programmers, for the reasons outiined above.

User interface design

Introduction

The most fundamental design principle is to *Know the User* (Hansen 1971). The designer should be intimately acquainted with the user’s needs, the user’s frame of reference and experience, and the conditions under which the tasks will be performed. Thus the first step in the design process is to find out about the user. The designer must also know about the operating characteristics of humans in general, particularly the fact that humans are fallible in both action and memory. They make all kinds of mistakes and often fail to remember things, especially detail, even when they know them quite well. The design must be created within this basic framework. The less the designer establishes and uses the framework, the more the resulting interface will be deficient in meeting the user’s needs and in providing a natural. comfortable, non-intrusive tool to help in solving the user’s problems.

The second major principle is to design the tools the user needs, fit for the user's tasks. If a system is designed that does not provide the specific task-oriented components needed by the user, in a suitable form, it is deficient in a very fundamental sense. Application of the first principle results, amongst other things, in a detailed statement of the specific tasks that the user must perform. The next step in the design process is to design the tools to meet each particular user need in the specific task context. The tools should be consistently integrated into an overall system that fits the users conception of the task and task environment. Perhaps the best approach to ensure this is done effectively is to start writing the user manual for the system at the earliest possible stage in the design process. certainly before starting to write code.

If possible, a dialogue prototyping aid should be used to help to formulate the design realistically, to allow the effect of design decisions to become obvious before they become cast in stone, and to give the user direct experience of parts of the finished dialogue. Dialogue prototyping aids such as *Flair* and *Ruby* (McLaughlin 1984), *RAPID/USE* (Wasserman 1985). or the *IDD* (Hill and Irving 1984), go beyond mere screen management in that they are designed to provide high level dialogue creation facilities, including ready-made screen formats, special facilities for handling input and output in a bullet-proof manner, and the like. Without such high-level aids, the user manual must serve both purposes. At the very least. diagrams and detailed dialogue scripts covering the kinds of interaction proposed should be prepared for inclusion in the user manual, which then also becomes the main framework for refinement of the design prior to implementation. Obviously, a detailed prototype is better for getting a feel for the system, but such material is required in the user manual anyway. Both a prototype and sample dialogs relate to Gaines' rule: introduce through experience (Gaines in press) and, whilst the prototype will be superseded, the sample dialogs will have ongoing value for reference and training.

If the designer tries to implement the system before writing the user manual the designer will not know what he or she is trying to design except in terms too general to serve as an adequate specification, and the user will have no basis for evaluation or communication prior to irrevocable commitment. The user manual is a fundamental part of the specification since it specifies many factors that are otherwise very hard to pin down. It specifies the purpose of the system, and how this purpose is to be accomplished in functional terms. It specifies what facilities will be available, how these facilities relate to the functions required, and how to access them. It specifies screen layouts, keyboard layouts, forms and the like. It explains how errors, backup, changes of mind and the like are handled. It helps to create and teach an appropriate model for the user. And, in the absence of a working prototype, it contains sample dialogues, so that inconsistencies and problems may become evident and be resolved before implementation starts. Finally it points out the limitations of the design, to serve as a warning, and as a basis for future extensions. Thus the user manual documents the formal selection and instantiation of rules, derived from the design principles, that detail the presentation level, and tell implementers exactiy what is required. Formal specifications, in the usual sense, can only safely be created once the initial user manual is complete and has been discussed with a reasonably diverse sample of users (even If they are programmers!)

The design process

In the design process. one must establish the goal to be achieved (the creation of some kind of product which meets functional and aesthetic criteria). Broad

principles, appropriate to the overall design, are adduced. These principles are successively refined and fleshed out until detailed design rules exist that are suited to controlling the implementation of each module of the final product, as well as the integration of modules into the desired complete system. In many disciplines, there is considerable previous experience in design and implementation for the various products associated with the discipline. This is an important part of the methodology of that discipline. Thus the procedures for building the usual kinds of bridges are well established, and encompass such things as the scientific basis for calculating strength and loading, engineering practice for: different kinds of design; safety; maintenance; and appearance. The procedures include high level principles relating to function and appearance (e.g. all bridges should last indefinitely, carry certain static and dynamic loads, and blend into their environment), as well as specific rules for achieving certain ends (such as a requirement for certain materials to be used in foundation structures if the soil acldlty exceeds a certain level; and for restricted ranges of paint colour). This body of methodology evolves, and is continually updated to reflect experience, improvements in technology, new knowledge, and the fashions of the day. The detailed rules tend to depend closely on the local conditions, and the specific service or facility to be supplied. They tend to be less easily generalised. This is why they are called rules rather than principles. Thus, one may have a principle in traffic engineering. that vehicles moving in opposite directions should not occupy the same piece of paving. A rule would then be formulated that would constrain all traffic to keep to the right side of the road in the direction of travel. Given common sense (e.g. the roadway must be wide enough), such a rule is easily enough implemented by the designers as part of the users' context. Often, transforming principles into rules detailed enough to be used directly, and selecting those that apply in a given case involves the services of a trained professional, who will not only possess knowledge and experience. *but will make an intensive study of the particular problem*. Carroll (1983) has referred to *architectural form* and presentation in interface design, noting that form can be characterised by general principles whilst presentation cannot, being determined by the detailed requirements of the particular interface. This follows from the discussion above, with presentation involving detail that depends on both changing technology and informed selection amongst detailed implementation rules. It is keeping up-to-date, knowing the rules, and making selections appropriate to doing the job well, that provides designers and architects with well paid employment. The rules and selections made embody experience and skill in following the principles—which specify the goals to be achieved in design in order to ensure *Accessibility, Productivity, Reliability, and User satisfaction*.

Computer science is such a new and rapidly expanding discipline that design methodology and the required basic knowledge is fragmentary and incomplete, especially in the most recent and more interdisciplinary areas. The human-computer interface is one such area.

Although there is a lack of design methodology for the human-computer interface, there is a considerable and growing body of literature that is relevant. It is the definition of principles and their refinement into design rules, based on this knowledge, as well as the extension of this body of knowledge, that forms the subject of research on the human-computer interface. Much of the research activity must be based on experiment, particularly evaluation of alternatives. It is very difficult to create and control experiments on human-computer interaction. There is very little previous knowledge on which to draw, and it is the integration of human and computer skills together with subtle forms of communication by images and language that is at issue. This situation presents a formidable barrier to progress. Moran (1981) has highlighted the problem:

> There are at present no systematic methods or body of knowledge to help a designer accomplish [his or her] goals for the interface—it is a purely intuitive endeavour.

though possibly he goes a little too far.

The basic tenet of Moran's paper is that in order for users to be able to use a system. they must have a model of the system; that learning to use a system consists of building the required model to use it; that any aspect of the system that enters the model is part of the user interface; and, thus. that designing the user interface is tantamount to designing the user's model of the system, and hence designing the system itself. If the user's model has elements that conflict with any previous experience and training in accomplishing tasks, or that are difficult to deal with for some other reason, performance will be degraded, and a potent and unnecessary source of errors wll have been created.

The main thrust of Moran's paper is to attempt to define a representational framework within which to design interactive systems, starting from a specification of tasks and abstract concepts (the *Conceptual Component*) and proceeding, by a process analogous to stepwise refinement, to the *Interaction Level*. However, it only constitutes a formalism for specification and does not talk about the goals of the design process, or their rationalisation. The framework is as yet incomplete, and avoids discussion of interaction level details (corresponding to Carroll's *Presentation*). Whilst offering a potentially valuable tool with which to control the design process (Augmented Transition Networks are another attractive possibility (Jacob 1983. 1985), the paper does not address the question of what principles the designer must satisfy in creating the design, nor what detailed rules might begin to embody those principles. It is with such principles, their derivation, and their rationalisation, that we are presently concerned. Although these too are still undergoing revision and re-formulation, it is believed that the principles and rules are specific enough to serve as a partial basis for rational design of the human-computer interface, regardless of the formal framework used to describe the interface. This latter question, what alternatives may exist for formal description, is not considered in the present paper.

A final point needs to be made, in the context of the design process. The current state of knowledge in human computer interface design is such that the designer frequently lacks the primary data needed for design. This can mean that experiments to gather data are be needed, and the design process becomes a research project. It may not even be clear what questions to ask, as a basis for research. Stuart Card, at Xerox PARe, suggests an effective approach to one component of this problem, namely *key factor analysis*. By this. he means that the designer or researcher must go into a knowledge based—rather than rule based— problem solving mode and identify the key factors in a design[1]. Thus. for an interactive system based on multiple windows, a key factor is likely to be screen "real estate". It limits the number of windows and/or the size of windows that may be viewed concurrently. The designer is then forced to consider what must be viewed simultaneously (non-overlapping windows), and what can be viewed sequentially (overlapping or stacked windows, or even serial screens within windows). This then raises the questions of: "How does the user operate? What are the cognitive structures involved in the user's approach to carrying out the task(s)?". Tasks that depend on sequential operations can use the latter type of window arrangements whilst the others demand tiled (non-overlapped) windows. From this, the minimum screen size can be determined, and, if necessary, the screen can be made larger or, failing that possibility, the need for task redesign is made clear. It is this kind of problem that justifies Moran's statement. However, although design is still a combination of art with ill-defined science, half a loaf is better than no loaf. No apology is made for presenting principles and techniques for design in as much detail as is reasonable and possible despite their limitations.

---

[1] Knowledge-based performance requires explicit knowledge of how the world works. This means it needs accurate dynamic, generative models of extensive parts of the real world and the ability to apply them. Significant knowledge-based performance has yet to be achieved by computers.

Principles for human-computer interface design

In what follows, numbered bold items indicate selected principles for design, items (1) and (2) being of pre-emptive importance. Under each numbered category elaboration of the principles occurs, and curly-bracketted entries '{ ... }' indicate, in note form, the relevant human andlor machine consideration(s) involved. Quotes from other sources and restatements of material above appear in smaller print.

Like many categorisations this one is probably somewhat arbitrary. Other divisions are undoubtedly possible and the inevitable overlap and uncertainty between the categories makes them less clear-cut than one would like. However, the selection and structuring does attempt another small step towards the goals of formalising the user interface design process; and giving more detailed guidance on the "whats" and "whys" of the process. It does not attempt to provide the kind of formal framework for specification aimed at by Moran (1981)—as noted, but it does provide a structured guide concerning what goal-oriented content should be fitted within such a framework.

The first two principles have already been considered in fair detail as part of the overview, so that they are somewhat summarised in the categorisation that now follows.

(1)-Know the user.

> The designer should be intimately acquainted with the user's needs, the user's frame of reference and experience, and the conditions under which the task (s) will be performed. The less this is satisfied, the more likely it is that the interface design will be deficient in meeting those needs, and in providing a natural, comfortable, non-intrusive tool. Knowing the user also includes knowing about the limitations, strengths, skills, and characteristics of humans in general.

Investigate the characteristics of the user population directly (not second-hand: stereotypes[1], percentile measures, conceptual framework … .) and design accordingly. The designer must remember that he or she is not necessarily a good example of a typical user, even if there is considerable overlap in terms of experience and task characteristics with some users. Self-assessment of interface components can often short-circuit the need for extensive human factors experiments, and guide design, but can be very dangerous if carried too far. Presumably every system is 'friendly' to its designer.

Thus the designer must appreciate that humans vary greatly in their physical and mental characteristics, being especially aware that 'different' does not usually imply 'inferior'. People have valid preferences that often reflect the very experience that the designer should be exploiting. There are a number of important aspects of the human operating characteristic, including the fact that mistakes are inevitable, whilst fatigue, boredom, panic and frustration cannot rationally be condemned, but only avoided by careful task and interface design. Humans also have strengths and abilities denied to machines—that is why they are designed into the system in the first place—and the designer should build on these qualities, whilst designing to overcome the less convenient human characteristics. Indeed, learning to use neutral terms to discuss such problems, rather than using terms such as 'weakness'. 'failure', 'operator error', 'impatience', and the like, is probably half the battle in meeting these aspects of the design goals.

---

[1] In this context, the term "stereotype" is not derogatory, but represents the expected behaviour of various classes of user.

(2)-Design the tools the user needs. fit for the user's tasks.

> If a system is designed that does not provide the specific task-oriented components needed by the user, in suitable form, then it is deficient in a very fundamental sense. Application of the first principle results, amongst other things, in a detailed statement of the specific tasks that the user must perform. The next step in the design process is to design the tools to meet each particular user need in the specific task context The tools should be consistently integrated into an overall system that fits the user's conception of the task and task environment Perhaps the best approach to ensure this is done effectively is to start writing the user manual at the earliest possible stage in the design process. If possible, a dialog prototyping aid should be used to help formulate the design realistically, to allow the effect of design decisions to become obvious before they become cast in stone, and to give the user direct experience [promoting user involvement in the design process].

A reasonable outline for such a manual would be:

Summary (Briefly summarises the content of the document)
Introduction (Gives background to the application. What is the general area into which the task falls. Who are the users and what are their overall needs. Who was consulted. Any special circumstances or difficulties …)
Purpose of the system (The specific purpose of this interactive dialog—what job does it do for the user. Be brief. Details can go in the section on capabilities or transaction details.)
System overview and rationale (An outline view of how the system is organised, and why it is organised this way. A diagram showing the main blocks, paths and relations may help here.)
System capabilities (A list of the specific capabilities of the system, rather than just a statement of the overall function as stated in the "Purpose" section.)
Transaction details (This, and the next three sections, are the most detailed part of the user manual. An introductory subsection should explain overall screen formats and the like; and then the actual screen formats, error handling, default entries, form of feedback, any other keystroke saving facilities [e.g. menu short-cuts], and so on should be outlined for each transaction. Common features [such as editing modes] go in the introductory subsection to this section.)
Help facilities available (How help is organised and how to access it; including help-help)
Facilities for audit and gripes (How they are organised and how to access them.)
Sample dialogues (A few representative samples of typical dialogues, using reasonably exact representations. The main point is to give an idea of the system in use, as opposed to the previous section which tends to give a picture of the parts, without relating them. The overview diagram from the overview section may be a useful aid in explaining how dialog sequences work out.)
Critical review of system with a note of 'next-release' improvements (Stand back and try to point out any problems with the system. This section provides a guide for anyone who might have to produce the next release.)
Acknowledgements
References
Appendices (The most likely items here are forms associated with some existing system.l)

The remaining principles, which serve as categories within which design rules can be successively refined, really follow from the two major principles above, and cannot be applied in a vacuum. The designer must have a goal, a functional context, and an understanding of the available materials, in order to apply principles and instantiate design rules in an effective and integrated manner.

(3)-Make the system easy to learn and remember

(See, also, section (8))

Keep the system as simple as possible whilst still meeting other criteria. Ask what can safely be excluded, not what might be put in.

{Short-term memory is limited—magic number 7±2}
{Too much detail confuses, and slows human operations}

Be both consistent and uniform in the style and details of the dialogue. As Gaines has succinctly stated (Gaines in press):

> All terminology and operational procedures should be uniformly available and consistently applied throughout all system activities.

Use familiar terms and familiar concepts.

{Humans learn new skills in terms of past experience}

{Human memory is associative}

{Negative transfer occurs between incompatible skills}

Use mnemonic symbols for all things symbolised (icons can also be mnemonic symbols).
(**Mnemonic symbol: easily remembered symbol having strong association to the item symbolised**)

{Associative character of human memory}

Facilitate the formation and use of models: the user should be taught an adequate model of the system and the system should acquire and use information about the user and his or her goals. The user's mental model is central to this. It constrains the design in the first instance and serves as a major link with the user in learning, using and improving the dialogue.

{Humans need structure to combat complexity}

{Humans work best in terms of their own conceptual models}

Maximise continuity in all aspects of the interaction: visual, tactile, contextual, command language use, layout. stereotypes developed … (c.f. Foley & Wallace 1974)

{Physically and perceptually obvious}

{Models, and transfer of training}

Provide excellent help facilities. Help should be specific to the context where it is needed. The user should not have to work out how to access information on something he or she doesn't understand. It should be invoked by some obvious action, probably even automatically in certain cases. Initial help (especially if automatic) should be succinct. Repetitive requests should invoke increasingly detailed help (Gaines (in press) "*Query-in-Depth"*). The user may quite possibly need help with the help facility itself, to avoid interactive deadlock, if only an expandable index, richly laced with synonyms (a "*Help-Help"* facility).

{Well designed help is a powerful learning aid}

{Human memory is fallible}

{Don't give a person a job if the job can be defined so a machine is better}

{Even experts cannot always remember everything}

Optimise learning/skill-acquisition; provide specific aids to learning. Note that learning takes time to happen and requires support and focus.

{Humans are adaptable, but also need to learn to use the system}

Understand what the human users are trying to do in their terms and try to help them do it

{Humans work best in terms of their own conceptual models}

Present system functions and facilities in a form appropriate to the user's skills and background. Remember, too, that even experts have to learn, may forget, and also evolve in terms of the use they make of any system, so that, even for experts, easy learning must be provided. In any case, experts can get quite frustrated when they are not on their home system. Indeed, this may explain why "experts" tend to be so partisan about the operating systems and languages they use.

(4)-Deal with errors in a positive and helpful manner

(See, also, section (8))

Avoid errors if at all possible, without restricting the user. Thus a menu prevents invalid commands, but may restrict an expert (provide alternatives).

{Mistakes are an inevitable accompaniment to human performance: the more errors are possible, the more will be made}

Provide assertive explicit error messages that identify the cause of problems and set users on the right track to correct them (c.f. Smith 1975; an “assertlve” message carries no emotional overtones—is not patronising, obsequious, or rude).

{Humans make errors}

{Humans need to maintain a good self image}

{Humans do not perform well if they do not feel in control}

Provide “reasonableness checks” on input data. and correct “obvious” errors automatically, normally notifying the user (unless the user explicitly *chooses* to disable notification).

{Humans make errors}

{Humans should not have to do things the machine can do well}

Don’t make the user feel stupid or put down by a literal. unchecked interpretation of user input when the resulting action would be inappropriate, or would have serious consequences.

{Maintain a good self-image for the user}

{The machine should be a “good servant”}

(5)-Consider. protect against and. if possible avoid both human and machine failure modes

Make the system “bullet-proot”, and control exit from the dialogue to other host facilities

{Humans are deranged by unexpected action}

{The computer should be a ‘good servant’}

Allow forestalling of prompts. That is, allow type-ahead. Then, if the user has anticipated a prompt and given the response already. drop the prompt. This can automatically provide for shortcutting menu hierarchy traversal, and is of especial importance when speech output is used.

{Humans do not function well when subjected to delays}

Make simple repetitive tasks the responsibility of the machine

{The machine is tireless, good at repetition, and direct memory tasks; humans are not}

Aim for optimum stress

{Humans become deranged if bored or overloaded}

Provide back-up for machine components in human-computer systems, especially critical real-time systems (of course, human back-up may also be needed)

{Machines tend to fail suddenly and completely}

{Machine failure is very likely to overload the human capacity,at least in the short term}

{Humans may panic when subjected to unexpected emergencies}

Be forgiving and flexible. Support not punish.

{Humans do not handle rigid protocols well}

{Humans become deranged by perceived lack of cooperation}

Provide variety to motivate and interest the user (though this conflicts with consistency and is secondary; initiative on the part of the computer frightens some and stimulates others)

{Humans enjoy variety. It motivates them provided that they still feel in control}

(6)-Provide good feedback on what the system is doing. and how this fits in with the overall structure of the system

Provide clear feedback to the user, especially for errors, but always so that the user is sure of what is going on

{People perform better, learn better, feel more in control, and develop a better model of the system with feedback, which increases motivation, builds confidence, and promotes accuracy}

(7)-Structure the interaction, the information presented or requested, and any data used, to help the user cope with its complexity

Break up any information for recall or choice into familiar groupings

- · familiar organisations, e.g. mimic diagrams, relate closely to a user's reality
- · exploit 'chunking'* and association, especially in choosing mnemonics
- · pictures are worth many words
- · layout, fonts, colour. ...
- · hierarchical or other *appropriate* organisational structure

{Short-term memory characteristics}
{Humans find unstructured detail difficult to manage}

Provide structural cues to combat complexity
{Don't leave the human to work out something if the machine can help}

(8)-Convey a real feeling of control to the user, even when it is the computer that is asking the questions

Make it easy to 'escape' and to correct errors. Confirm actions having major consequences, esepcially if recovery is not practical. People often don't appreciate the consequences of their action until too late, so that an *UNDO* command is a valuable aproach to this problem. A system that is uniform, consistent, and easy to learn and remember promotes a feeling of control (see (3) above).

{Humans do not function well if they do not feel in control}
{Humans do not feel in control if they find it impossible to escape from a course of action, or reverse the effects of a mistake}
{Humans make errors, it is part of their operational nature and designers must design on that basis making allowance for the inevitable. Note: this actually reduces errors because of reduced anxiety, apart from the direct effect on productivity}
{Humans will learn more quickly and effectively if it is easy to experiment without horrible consequences}

Make provision for the user to tailor the interaction to his or her needs
{The user must feel in control and be in control}
{Individual differences—people vary}

Minimise the activity needed to initiate actions—but don't penalise verbosity.
{Humans do not feel in control if control requires great effort}

Don't have the machine do unexpected or unreasonable things
{Humans are good at initiative and creativity; if the machine competes in this, panic may ensue, the image of the machine as a good servant is destroyed, the user's model is disrupted, and the user no longer feels in control}

(9)-Consider carefully the division of labour in allocating tasks between human and computer

Capitalise on the strengths, and avoid the less convenient characteristics of both humans and machines. For example, divide tasks and present information in a way that allows the human to exercise integrative skills effectively (e.g. by pictures that reveal matters of importance, rather than tables of numbers) (for example, see Tufte 1983)

{This is the prime reason for having the human acting in co-operation with the system to start with!}
{Human abilities and machine abilities are complementary}

Minimise keystrokes within the constraints of familiar mnemonic, redundant coding for all symbolisation. For input, function keys can be used, whilst also allowing multicharacter symbols. Voice buttons that work provide an ideal access to functions, except they are not

* "Chunks" are groupings of things that are familiar to people so that they treat them as one item (Miller 1956)

self-documenting. Providing sensible defaults for input (including commands), wherever practical, is another excellent measure that can be taken.

{Physically obvious; don't have the human doing unnecessary work}

{Function keys aid memory, as well as reducing keystrokes and promoting association}

{Less chance for errors}

Provide for the machine to do all arithmetic—either automatically, or by a calculator function

{Humans are slow and inaccurate at arithmetic}

{Use of a real calculator breaks continuity}

Use documents, where they perform best for people

{Humans scan documents easily and find them more convenient than screens for some purposes such as browsing and searching for answers to ill-defined requests, or reading in the bath}

(10)-Take into account human performance limits

Present information legibly

{Human performance limits}

Don't overload the human communication capacity

{Not only straight information loss, but also the multiplicative effect of confusion}

Allow enough time for the human to function

{Humans have limits to their information processing capacity}

Consider carefully the arrangement of the physical workplace

{Humans have limits to their physical reach, direction of gaze, andtheir ability to see when subject to glare or reflections …}

(11)-Ensure that the user is psychologically comfortable

Provide adequate rest periods for human operators

{Humans require rest and recreation to maintain motivation and performance}

Make the system aesthetically pleasing

{Humans perform better when they feel someone takes an interest in their working conditions}

Don't 'put down' the user

{Humans need a good self-image to operate properly}

(12)-Ensure that the user is physically comfortable

Arrange a comfortable workstation

{Discomfort leads to fatigue and distraction, hence to errors and lowered productivity}

(13)-Design is an ongoing process. Provide facilities to monitor overall system activity

Provide an audit trail for tracking down problems, as well as for security

Log system activities (this requires careful selection and processing of important. necessary material, plus the discard of all else. or there will be an unreasonable amount of unstructured data; it is not reasonable to expect this kind of log to be a substitute for carefully designed experimental data collection intended to support the testing of hypotheses in evaluation trials; it can only act as a focussing mechanism)

Make explicit provision for user complaints (a “gripe” facility), that is easy to use, as a dialog excursion. You need the information for debugging and development.

{Don’t ask the human to do alone what the machine can help with; reduced effort increases response likelihood}

{Humans perform better when they feel someone takes an interest in their working conditions}

References

BCS (1977) A series of news reports on July 7th. (p1 & 3). August 4th. and September 29th. on a major British Motorway disaster. *Computing* **5**

BERGLAND. G.D. (1981) A guided tour of program design methodologies. *Computer* **14** (10), 13-37, October.

CARROLL, J.M. (1983) Presentation and form in user interface architecture. *Byte* **8** (12), 113-122, December.

CARROLL, J.M. & THOMAS, J.C. (1982) Metaphor and the representation of computing systems. *IEEE Transactions on Systems, Man and Cybernetics* **SMC-12**, (12), March/April.

CINCOM (1981) MANTIS: an application development system. *Technical Brochure GSS-31-2/81*, Cincinatti: Cincom Systems Inc., 2300 Montana Avenue, Ohio 45211.

DAVIS, A. (1984) (Personal communication. Davis is working on 5th. Generation Computer Architecture at FLAIR. the Fairchild research laboratory in Palo Alto).

FOLEY, J.D. & WALLACE, V.L. (1974) The art of natural graphic man-machine conversation. *Proceedings of the IEEE* **62** (4), 462-471 April.

GAINES, B.R. (in press) *Dialog Engineering*. Computers and People Series. New York: Academic Press.

HANSEN. W.J. (1971) User engineering principles for interactive systems. Fall Joint Computer Conference. *AFIPS Conference Proceedings* **39**, 523-532, New York: American Federation for Information Processing.

HILL. D.R. & IRVING. G. (] 983) The Interactive Dialogue Driver: a UNIX tool. *Proceedings CIPS Session 84*. Calgary, May 9-11, Toronto: Canadian Information Processing Society.

JACOB. R.J.K. (1983) Using formal specifications in the design of a human-computer interface. *Communications of the ACM* **26** (4), 259-264, April.

MACGUIRE, M. (1982) An evaluation of published recommendations on the design of man-computer dialogues. *International Journal of Man-Machine Studies* **16** (3), 237-262, April.

McLAUGHLIN, L. (1984) Rapid prototyping. *Proceedings CIPS Session 84*, Calgary, May 9-11. Toronto: Canadian Information Processing Society.

MCLAUGHLIN, L. (1985) Private communication.

MILLER, G.A. (1956) The Magical Number Seven, Plus or Minus Two: Some Limits on our Capacity for Processing Information. *Psychological Review*, **63**, 81-97.

MORAN, T.P. (1981) The command language grammar. *International Journal of Man-Machine Studies* **15** (1), 3-50, July.

NORMAN. D.A. (in press) The truth about UNIX: the user interface is horrid. *Datamation*

NORMAN. D.A. (1983) Design principles for human-computer interfaces. *Human Factors in Computing Systems—CHI '83 Conference Proceedings*, Boston, December, Baltimore: ACM (PO Box 64145).

PARNAS, D. (1984) Personal communication.

PERLMAN, G. (1981) Two papers in cognitive engineering: The design of an interface to a programming system, and: MENUNIX: a menu-based interface to UNIX (User manual). *Center for Human Information Processing Report No. 8105*, La Jolla, California: UCSD.

PERLMAN, G. (1984) Natural artificial languages: low-level processes. *International Journal of Man-Machine Studies,* **20** (3)

SHEIL. B.A. (1981) The psychological study of programming. *Computing Surveys* **13** (1), 101-120. March.

SMITH. M.J. (1975) *When I say no, I feel guilty*. Bantam Books: Toronto, 324pp.

TUFTE, E.R (1983) *The Visual Display of Quantitative Information.* Graphics Press: Box 430, Chesire, CT 06410, ISBN 978-0961-39214-7

WASSERMAN, A.E. (1983) The Unified Support Environment: tool support for the user software engineering methodology. *Proc. IEEE Computer Society SoftFair Conference*, July

WILLIAMS. G. (1983) The Lisa computer system. *Byte,* **8** (2), 33-50, February.

WITTEN, I.H., BIRTWISTLE, G.M ., CLEARY, J., HILL, D.R., LEVINSON, D.L., LOMOW, G ., NEAL R ., PETERSON, M ., UNGER, B.W ., & WYVILL, B. (1984) Jade: a distributed software prototyping environment. *Proc. 1984 SCS Multiconference.* San Diego. February.

drh