1. Dialog engineering

B. R. GAINES AND M. L. G. SHAW

Department of Computer Science, York University, Downsview, Ontario, Canada

I. Introduction

Thirty years ago a technological revolution took shape as the first stored-program computers stumbled through their early calculations. Fifteen years later computer systems had become sufficiently reliable for a second revolution to take place as the first time-shared *interactive* systems began to offer their services. The MIT MAC system (Fano, 1965) in 1963, the RAND JOSS system in 1963–1964 (Shaw, 1968) and the Dartmouth College BASIC system in 1964 (Danver and Nevison, 1969), pioneered a new style of computing in which users at remote teleprinter terminals were in direct communication with the computer. No longer did the user have to prepare all his programs and data meticulously in every detail in advance for submission to a queue of jobs eagerly awaiting the magic moments when the computer was actually working. Titles of books in the mid-1960s give some indication of the dramatic impact of this *Transition to On-Line Computing* (Gruenberger, 1967) through *The Challenge of the Computer Utility* (Parkhill, 1966) using *Conversational Computers* (Orr, 1968) to offer *Computer Augmentation of Human Reasoning* (Sass and Wilkinson, 1965). However, for many years to follow the majority of users still continued to use batch-processing systems.

Now, another fifteen years on, the continuing decline in computer costs and increase in reliability has made hands-on, interactive use of computers a commercial reality for all users (Gaines, 1978a). Conversational computing will become increasingly the normal mode of operation for the majority of computer users. We have previously emphasized the important role of *personal computers* not just as a change of technology but as a new force in the relationship between the individual and society (Shaw and Gaines,

DESIGNING FOR HUMAN–COMPUTER COMMUNICATION
ISBN 0 12 643820 X

Copyright © 1983 by Academic Press London
All rights of reproduction in any form reserved

1981b); a point argued more dramatically by both De Bono (1979) in *Future Positive*, and Toffler (1980) in *The Third Wave*. This presupposes the possibility of a close relationship developing between people and computers akin to that which Licklider (1968) calls *man-computer symbiosis*.

Despite the recognized importance of this transition to *conversational* and *personal* computing the study of the design of man–machine dialogs has only recently gained recognition as a field for the development of computer science and technology. It is interesting to examine the history of this growth of interest in the user interface at a cognitive rather than solely ergonomic level. The Klerer and Reinfelds (1968) collection of some 45 papers on *Interactive Systems for Experimental Applied Mathematics* already contains studies such as Fried's (1968) *On the user's point of view* emphasizing "direct user control" and "development of individual user systems", and Patton's (1968) on *A message system for interactive dialog* emphasizing a state-determined branching dialog structure as a standard interface to all interactive programs. Hansen (1971) at the Fall Joint Computer Conference appears to have made the first attempt to tabulate some *user engineering principles* for the design of interactive systems.

The key significance of the growing community of non-computer-oriented users of interactive systems was emphasized by Wasserman (1973) at the National Computer Conference two years later in his paper entitled *The design of idiot-proof systems*. The search for a more pleasant terminology for this key class of people, the *non-professional*, *naive* or *casual* users, continues today. They are only thus in relation to their technical knowledge of the inner workings of the computer and the need for such terminology is a significant message about the history of computers: the early users were necessarily computer technicians; the next generation of users had to go through these technicians in order to access the computer (Gaines, 1978e); and it is only now that computer usage has become potentially completely open. The motorist is not non-professional nor naive because he does not understand the internal combustion engine and the motor trade takes pride, and profit, in making him an expert user without needing to have that knowledge. The computer trade will only attain its full potential when there is similar pride, and profit, in making computer power readily available to users whose expertise is in their usage, not in their knowledge of bits, interrupts and DO-loops.

The first book on the topic of dialog programming was Martin's *Design of Man–Computer Dialogues* in 1973, followed in 1977 by Gilb and Weinberg's *Humanized Input*. Both books give a wealth of anecdotal experience and a variety of recommended techniques, but neither was then in a position to give a systematic account of what constitutes good dialog programming. In the mid-1970s we were in the same position with

programming man–computer interaction as we were with hardware design 30 years ago and software design 10 years ago—we were reliant on unsystematized *craft* skills passed on erratically through fortunate *apprenticeships*. Through evolution many systems had become reasonably satisfactory in their basic dialogs and these could be copied in similar situations, but we had no production techniques for *dialog engineering* on a specified, scheduled, quality-controlled, reliable basis. The effects of this were starkly stated at the 1974 National Computer Conference:

> the lowered cost of computer access and the proliferation of on-line systems produced a new breed of users, people whose expertise was in some area other than computer technology. As their initial fascination with conversational computing wore off, users reported experiencing feelings of intense frustration and of being 'manipulated' by a seemingly unyielding, rigid, intolerant dialogue partner, and these users began disconnecting from time-sharing services at a rate which was very alarming to the industry. (Walther and O'Neil, 1974)

These remarks were still echoed by Nickerson (1981) recently in his paper *Why interactive computers are sometimes not used by people who might benefit from them.*

In the mid-1970s there was an increase in studies of man–computer interaction commencing with attempts to give: rules for "well-behaved" interactive procedures by Kennedy (1974); a systematic *Dialog specification procedure* by Pew and Rollins (1975) at Bolt Beranek and Newman; and systemic guidelines with theoretical and psychological underpinning by Gaines and Facey (1975). Thereafter the literature on man–computer dialog design grew rapidly and began to form the basis of overall appraisals such as that in Shneiderman's (1980) book on *Software Psychology* where the work already mentioned, together with that of Cheriton (1976), Gebhardt and Stellmacher (1978) and Turoff *et al.* (1978) is tabulated and discussed. Various collections of papers have also been published recently on *Human Interaction with Computers* (Smith and Green, 1980), *Methodology of Interaction* (Guedj *et al.*, 1980), *Man–Computer Interaction* (Shackel, 1981) and *Computing Skills and the User Interface* (Coombs and Alty, 1981). It is often forgotten that Petri's (1962) work on net representations of concurrent systems arose from his studies of the dynamics of complete systems including the users. Dehning *et al.* (1981) have recently published a wide-ranging presentation of *User requirements in dialogs* based on petrinet concepts.

Naturally the many sources of guidelines are not always mutually consistent and Maguire (1982) has given a coherent account of them and suggested resolutions of conflicting guidelines. Some of the apparent

inconsistencies are themselves significant—we have suggested elsewhere (Gaines, 1980) that "the opposite of a statement worth making is also a statement worth making". There is a further analogy from the motor trade where the instructor will initially tell a trainee driver to "let in the clutch very gently" in order to avoid stalling the engine. Later, when the semi-experienced trainee is letting the engine race excessively, the instructor will tell him to "let in the clutch rapidly". These guidelines apparently conflict only if they are taken as universal generalities rather than rules appropriate to different stages of knowledge of the trainee. It is this discrimination and refinement in the engineering of man–computer interaction that we can see becoming significant in the 1980s.

In reviewing the next stages of development in man–computer communication (Gaines, 1978e), we stressed the need to consider both the internal models of the *worlds* within man and computer when setting up and analysing communication, and also the mutual models that the computer and man have of one another (Shaw and Gaines, 1981a). We have also emphasized the short-term practical role that developing a *technology of interaction* has for business applications of minicomputers in the next decade, and placed a *dialog processing module* on par with other fundamental modules for *database control*, *document production* and so on (Gaines, 1978a). This chapter may be seen as a more specific attempt to take that framework of mutual modelling and generate within it detailed characteristics for dialog engineering.

It would be rash to pretend any work at this stage can approach a complete formulation of the techniques for *dialog engineering*. In particular, the rapid change in interface technology requires constant re-interpretation of the guidelines, possibly even their fundamental modification if they are not systematically stated. The advent of *intelligent terminals* in which a local microprocessor gives very flexible control over a visual display, the availability of graphics as well as textual display, the capability to recognize handwriting and, perhaps soon, speech—all of these change the man–computer interface and require a re-appraisal of dialog engineering techniques. However, the rules given in this chapter do represent some further steps in the direction taken by the work noted above in beginning to systematize the rules for programming man–machine interaction through the engineering of formal dialogs.

II. Programming interaction

First let us make it clear what kinds of conversation and dialog are under consideration in this chapter. Natural language dialog with computers has

been the goal of much *artificial intelligence* research, and Weizenbaum's (1967) ELIZA with its *cocktail party* capability to maintain a conversation *without understanding*, and Winograd's (1972) SHRDLU that used its *understanding* of a narrow domain to allow facile manipulation of language, both gave hope that the goal was achievable. Petrick's (1976) survey of work on natural language-based systems demonstrates the increasing capabilities of such systems, and database query systems such as Hendrix's (1977) LIFER and Harris's (1977) ROBOT are very impressive in their abilities to maintain meaningful and useful dialogs with a wide range of real users. Indeed, natural language communication itself can now be seen as available technology limited in its application by more fundamental defects in our database systems (Gaines, 1979).

However, there are a number of reasons why such truly *natural language* systems will probably not play major commercial roles for some years to come:

(1) They still require large programs in powerful machines for their operation.
(2) The typing by the user to generate complete English sentences is too great—one day speech recognition (Cox and Martin, 1975; Woods *et al.*, 1976) may change this, but for the moment *unnatural* dialogs minimizing user typing activities are essential.
(3) In many applications where the computer replaces or mimics the clerk in an office the *natural* dialog is already highly formalized because it has evolved to carry information clearly and concisely—we have a formal *battle language* that is purely functional, resistant to noise and corruption, and readily assimilated by newcomers.

It is these more formal dialogs with which we are concerned in this chapter—they form the major requirement in current commercial systems, but their programming is still too much a *hit and miss* affair. However, most of the points made are basic systemic and psychological requirements that carry over to the free form natural language dialog that may one day become possible and desired. Figure 1 gives a sample of the type of dialog under consideration—in this case a data entry sequence in a securities dealing system. The line numbers are for reference only—user input follows the last colon on each line, lines 4, 7 and 8 offer default options, line 12 is validation before entry.

Seven years ago we attempted to analyse lessons learnt in a wide range of experience of *Interactive system development and application* (Gaines and Facey, 1975) in a variety of medical, industrial and commercial projects. One section of this paper was entitled *The programming of interaction*, and at the second draft we decided to make it rather more pointed by phrasing

```
 1  GDR:E
 2  Item:S
 3  Stock number:85
 4  Full name: Treasury :Exchequer
 5  Issue date:6/7
 6  Type (SMLTN):S
 7  Dividends/year: 2 :
 8  Final div date : 1/1 :5/8
 9  Redemption years(s):81
10  Coupon:3 1/4
11  Number of calls: 0 :
12    85:Exchequer  3 1/4  1981  5/8  S  6-Jul-77 OK?:Y
13  GDR:
```

Fig. 1. Typical dialog sequence.

the lessons learnt as a set of eleven *rules*. Whilst tightly formulated *rules of dialog* were undoubtedly at that stage of the game still pseudo-science, it has been invaluable in further development of dialog programming techniques to have a set of well-defined and precise guidelines with which to work. Having a defined set of rules has already led to suggestions for further rules and amendments to the first ones in the light of others' experience and these were published by Gaines (1978b, 1981). In the next section further developments of these rules are reiterated and expanded with a brief comment on each, and later sections use them as a basis for formalizing dialog programming, and refine them further. To complete this section we wish to establish some very basic guidelines for the objectives of actually having man–computer interaction to start with, and for the relative roles of man and computer in such systems. This is clearly a fundamental aim of a book such as this, to establish why man should wish to communicate with the computer, and on what terms he does so.

There are three remarks that for us epitomize our objectives in designing interactive systems: James Martin in his book on the *Design of Man–Computer Dialogues* states:

> man must become the prime focus of system design. The computer is there to serve him, to obtain information for him and to help him do his job. The ease with which he communicates with it will determine the extent to which he uses it. Whether or not he uses it powerfully will depend upon the man–machine language available to him and how well he is able to understand it. (Martin, 1973, p. 3)

Walter Doherty, Manager of the in-house systems at IBM's Thomas J. Watson Research Laboratories, remarked in a seminar at Essex University in September 1977:

> The computer is a tool to extend man's memory and reasoning power—these are the key bases on which technical advances should be evaluated.

And John Cleary at EUROCOMP 74 in discussing the role of such new system concepts as the *virtual machine* said:

> The only virtual machine of interest to a user is one that has abolished all lower levels of system and presents an *understandable* and *sympathetic* face to its user.

Doherty's remark brings out the role of the computer as a *tool* aiding a person in coping with their mental workload—in an interactive system the machine should do all that is possible to minimize unnecessary activity on the part of the user—it has an *emancipatory* role (Gaines, 1978d,e). Cleary's remark emphasizes that the interactive interface seen by the user is actually to a *virtual machine*—one which Krammer (1980) has termed the *attention processor*. Too often dialog programming is treated in an ad hoc fashion that takes no account of the fact that what the user is seeing is the highest level in a hierarchy of virtual machines (Gaines, 1975, 1976c) and that the rigour with which we treat the definition of the lower levels (in terms of procedures, languages, instruction sets, microprograms, etc.) can, and should, be applied at the dialog level—that is one objective of this chapter. Cleary then goes on to say that the virtual machine should have an "understandable" and "sympathetic" face, two apparently non-technical terms which may however be given quite definite meaning in terms of the comprehensibility of the virtual machine and the availability of helpful aids to the user. Much of what follows is concerned precisely with these two requirements.

In the next section we review basic rules for dialog engineering and explain them systemically, psychologically and through examples. In the following section we extend them to more general dialogs, particularly those through visual display units with local intelligence and graphics. We continue with a discussion of language support and implementation requirements and techniques for these rules.

III. The basic rules

This section gives an extended set of dialog programming rules based on those in Gaines and Facey (1975). The rules are classified into four categories: those concerned with systems analysis and development; those concerned with the user's adaptation to the system; those concerned with minimizing the mental load on the user; and those concerned with error detection and correction.

A. Systems analysis and development

The first rule previously proposed is concerned with the initial design of the dialog. In performing the systems analysis that precedes dialog design it is easy to forget how little conception potential users have of the interactive use of computers if they have never experienced it:

> *Rule 1. Introduce through experience*: interactive systems should be experienced not talked about. Get prospective users onto a terminal on a related, or model system before discussing their expected relationship to their own system.

This first rule is worth emphasizing if only because it does not occur in other guidelines in the literature and is prior to the presuppositions they make. However, it highlights a major pitfall into which we will all occasionally fall since the phenomenon of assuming that what we personally know and have experienced is *obvious* is a common one for all human behaviour. We cannot understand the output of a speech synthesizer and cannot understand how others can do so. Suddenly it becomes clear and we are able to comprehend each word. Thereafter we are surprised when others have the same problem in understanding initially what is, for us, now transparently clear material. We have always to remember the need for experience and learning on the part of others before we can have a meaningful discussion with them about topics they have never previously encountered.

Our second rule relates back to Hansen's (1971) first principle, "know the user", and is really a warning to the systems analyst who attempts to impose a new framework on an already existent and satisfactory situation:

> *Rule 2. Use the user's model*: use a model of the activity being undertaken which corresponds to that of the user, and program the interactive dialog as if it were a conversation between two users mutually accepting this model.

This second rule may be seen as asking the systems analyst to listen and learn the user's vocabulary—not to impose a new language unnecessarily in an existing situation. We use the word *model* deliberately, however, to emphasize that there is rather more than a vocabulary involved. We have to consider the way in which the words are being *used* to achieve some desired change in the world. It is an action-orientated, Gricean view of language (Bennett, 1976) that we have to take in order to make all aspects of the dialog, its vocabulary, its sequence, its consequences, appear natural to the user.

In this respect, the axiomatic, conventialist psychology developed by Kelly (1055) in terms of *personal constructs* has a significant role to play in formalizing the processes of systems analysis for dialog engineering. As he notes:

> Man looks at his world through transparent patterns or templets which he creates then attempts to fit over the realities of which the world is composed (p. 8). . . . A person's processes, psychologically speaking, slip into the grooves which are cut out by the mechanisms he adopts for realizing his objectives. (p. 49)

It is the "grooves" created by the construct structures of the users determining their vocabulary in communicating with one another that we have to reproduce. Kelly himself developed a means for eliciting such construct structures through his *repertory grid*, and in recent years his technique has been automated and substantially extended by Shaw (1980, 1981) in the *PEGASUS* and *FOCUS* systems. We have linked these techniques elsewhere with Checkland's (1981) *soft systems methodology*, and shown how they may be applied in systems analysis (Shaw and Gaines, 1982). Techniques for the interactive elicitation of personal construct structures may now be seen as essential tools for the dialog engineer in his preliminary analysis of the activities and requirements of a new user community. There are fascinating possibilities also, not yet exploited, of building such construct-elicitation processes permanently into a system to expedite some of the continuing system development called for in Rule 3 below.

The first two rules can also be given more context by noting that, in some sense, "the computer is itself in the situation of a new person coming into the office and adjusting to its procedures" (Gaines *et al.*, 1976). Rule 1 then says that before you decide how to deploy someone with new ways and skills take the opportunity to become familiar with them, or someone similar. Rule 2 says that if you are a newcomer in an office then try and slip into the familiar ways and jargon expected in that office—the new colleague who tries to impose new ways of performing existent tasks and new vocabularies for old conversations will not be welcome.

This animistic view of the computer is a very useful one if not overdone—at times of doubt the question, "well, what would I do?", is a good one. Indeed, one early qualm about computer systems, that they would depersonalize tasks through their lack of personality, is very rapidly dispelled as soon as one notes that interactive programs *do* have personality—usually some partial replica of that of their designer. This is unfortunate if the designer has hidden sadistic tendencies that surface in his programs. More usually it is the erratic human nature of the designer that comes over, being extremely helpful at one moment and then forgetting the obvious the next. We have suggested (Gaines *et al.*, 1974) that the personality one should attempt to project is that of a *servant*—a very Dickensian one with wit and intelligence rather than servility, but a servant dedicated to his master's ends. This role may be an unfamiliar one

for the modern technologist but it provides a sure logical foundation for many of the essential features of a good interactive system.

The model of the interactive computer as a newcomer in an office also suggests the next rule, since one vital characteristic of a person is their *adaptability*. If you as a newcomer have behaviour patterns which do not fit well into those of the office then social pressures soon lead you to conform (or leave). The natural mechanism for such pressures to have effect is ready adaptability to criticism, overt or covert. If what you do is not liked then you adjust rapidly until you gain approval, often through mimicry. At the current state of the art it is clearly the system designer who has to provide the interactive system with its adaptability. Fortunately, an interactive system can, and should, also be interactively programmable at the same location, probably at the same terminal, as the user. This enables the designer to set up a three-part dialog between himself, computer and user in which the effects of changes in programming the interaction can be made immediately apparent to the user and his evaluation received:

> *Rule 3. Design never ceases*: use the interactive capabilities of the system for programming also, so as to close directly the adaptive loop between system and user through yourself as designer.

This is not intended to say that interactive dialog *design* should take place at a terminal. Like any programming activity there is a need for thoughtful preparation away from the machine, and the best mode of working depends on the programmer. It is certainly to say, however, that the skill of rapid system modification at an interactive terminal is a very valuable one that should be developed, particularly in the current context.

Clearly the designer cannot remain an integral part of the system forever—even if an attractive role, with the declining costs of hardware relative to liveware, it is too expensive an option for most customers. However, Rule 3 does really mean *never* and provision has to be made both for the designer to play a continuing, but decreasing role in the adaptation of the system to changing circumstances and changing users, and also for the reversal of roles that will take place once the computer is the *old hand* rather than the *newcomer*. When the computer system is established and accepted we do then have to make provision for it to evaluate its users' behaviour patterns and inform them of reasons for their difficulties. Fortunately, one of the easiest things to program an interactive system to do is to keep records of its own activities:

> *Rule 4. Log activities*: use the computer to maintain selective records of system and user activities and provide programs to analyse these in terms of, for example, errors, broken down by user and by dialog sequence.

One golden rule is that logging without purpose is useless—the analysis programs must be designed as part of the logging procedure and their applications defined as well as any other on the system. Unnecessary actions on the parts of users are one obvious activity to trace—deletions, line cancellations, transaction aborts, and so on. If one user is producing excessive quantities of these he clearly needs guidance. If one section of the dialog is producing excessive quantities it clearly needs re-design. Proper long-term quality control of this nature is rarely found in current systems and yet the benefits are obvious, both for short-term system improvement and for long-term development of improved design techniques. We suspect the lack of proper dialog instrumentation is not due nearly so much to the commercial pressures commonly adduced as to the lack of appropriate dialog technology.

It seems probable that the logging recommended here will be subject to legislative constraints under pressure from labour organizations. In so much as it generates information about individuals, it would seem simplest to deal with it by analogy with other records of personal information, i.e. generally to allow free access to the individual concerned. In many ways it is tragic that it should be necessary to take legislation that far, and it is symptomatic of the (proper) fear of the use of computers in what Habermas (1972) calls a *technical cognitive* mode. In practice, if we wish to shape the behaviour of cooperative users (and if we wish to make all users cooperative ones), the most effective technique is to use the computer in *emancipatory cognitive* mode to portray to a user the structure and consequences of his own behaviour (as done by Mulhall (1977) in his clinical studies of personal relationships). Again one can envisage that advanced dialog processing systems will provide facilities for users themselves to access analyses of their own behaviour. In human terms it is the ability to look up and ask, "how am I doing", that an effective logging system can provide.

B. User adaptation to the system

One of the advantages of close man–computer interaction is that one may program the system to minimize the mental workload on the man. In so doing one concentrates upon the weaknesses of the human mind, taking into account limited short-term memory capacity, inaccuracy of calculation, etc., and it is easy to begin to think of the computer as the senior partner which must somehow compensate for the inadequacies of its weaker partners. A natural development of this "senior" role is to attempt to program the system to identify the weaknesses of particular users and to call up compensating facilities to cope with them, e.g. more guidance in the dialog, closer error-checking, tutorial messages, and so on. This was, for

example, part of the philosophy of many of the early computer-aided instruction systems, where *individualized* instruction was intended to be generated dynamically by the computer through the infinite wisdom of its programmer. In later developments there has been a swing towards *learner-controlled instruction* (Bunderson, 1974), because the on-line identification of human characteristics required for computer-controlled individualization proved unrealistic to implement.

Part of the problem is that in concentrating on the weaknesses of the person we forget their strengths, one of which is the modelling of the environment. There is a general phenomenon at work here which is summed up in the following rule:

> *Rule 5. The user will model the system*: do not assume that the user is a passive static system to be controlled, modelled and directed by the computer. Evaluate all actions of the system in terms of their effect on an actively changing user who is attempting to comprehend the system.

People automatically form internal models of their environment regardless of their naivety or intelligence. The mind is all the time searching out patterns of cause and effect—Michotte (1963) has shown that much of this activity takes place at a fairly low level and, like the perception of optical illusions, it is not under conscious control. The models formed by users may vary widely and have many connotations that arise from their previous background experience and bear no direct relation to the system. Referring back to Cleary's remark quoted in section II, we may say that to make the system *understandable* is to maximize the possibility of the user forming, with the minimum of effort, a model of the system which aids his effective use of it. Recently there have been some interesting studies of user models in some computational situations (Card *et al.*, 1980; Mayer and Bayman, 1981; Young, 1981).

Another danger of attempting to program adaptation to the user within the system is that the user is not only coming to comprehend the system but also to adapt to it himself. We have a control-theoretic situation in which two coupled systems are each attempting to adapt to one another and instability may result:

> *Rule 6. User should dominate computer*: either the computer or the user should dominate interaction or there will be instability. If the computer is to dominate it must be programmed to, and have sufficient information to model the user. If the user is to dominate then the computer system must be simple to understand. At the present state of the art the user should dominate the system.

It is worth emphasizing the grounds for this rule which are not *soft* ones relating to the proper role of computers in society. The channel of

communication from person to computer just does not provide a sufficient flow of information for on-line identification of the characteristics of the person. As mentioned previously in Rule 4, it is possible to gather specific useful information about individual users if this is designed into the original system, but the time-constant of doing this will be too long for immediate action within the interactive dialog. A similar phenomenon has been noted in investigations of adaptive process control—the normal channels of control only rarely provide a sufficient information rate about the plant characteristics for on-line adaptation to be feasible. Recently there have been some careful studies of the design of self-adaptive dialog interfaces avoiding these problems (Edmonds, 1981; Innocent, 1982).

One key rule that we proposed previously that arises out of the requirement to make the computer system understandable to the user is:

> *Rule 7. Avoid acausality*: make the activity of the system a clear consequence of the user's actions.

This rule sums up a variety of phenomena in interactive dialog. The basis for it is one that we have developed in system theoretic terms in a number of publications (Gaines, 1976a,b, 1977): the identification of an acausal system (involving non-deterministic behaviour) is very much more difficult than that of a deterministic system; people seem to pre-suppose determinism in the world but one may show (Gaines, 1976a) that if this supposition is false they will generate highly elaborate, "superstitious" models. We have suggested this as an explanation of the relative difficulty of using a time-shared computer rather than a dedicated single-user system. The hesitancies of the shared system due to activities that are deliberately hidden from the user have no apparent cause but this does not mean that the user will not continue, both at conscious and unconscious levels, to invent one. The importance of hesitancy patterns in the perception of speech (Goldman-Eisler, 1961) suggests that use of such information is, like the Michotte illusions, *wired-in* to the person and unavoidable.

Since Miller (1968) first suggested that there might be an *optimum* non-zero response time there have been many empirical studies ranging from large-scale statistical analyses of productivity showing optimum response time to be zero (Thadhani, 1981), through multivariate studies of controlled response time situations (Goodman and Spence, 1981) to laboratory studies of rates of information presentation (Bevan, 1981). There is no clear-cut result from such studies but they suggest that the objective of having short response times which are consistently related to user activity is a desirable one.

All this emphasis on the system being static suggests that one must under-utilize the capabilities of the computer to offer different dialogs

under different circumstances, e.g. a highly tutorial dialog to the naive user but a rapid, terse dialog for the experienced user. This is not the case—just that such changes should be under user control. The next section contains some consideration of minimizing the effort by the user in exerting such control, but there is one simple technique that has proved very effective in practice that we have previously noted:

> *Rule 8. Parallel-sequential trade-off*: allow the user maximum flexibility to make his responses holistically (in parallel) or serially (in sequence) according to his wishes.

Figure 2 shows the same sequence as that in Fig. 1 but with the user grouping his responses into clusters rather than making one at a time in sequence. At line 1′, the user puts in what were previously three separate inputs, E, S and 85. The system finds it has the responses to what were previously prompts 2 and 3 as well as that to 1 (in Fig. 1), and goes straight on to prompt 4′. Similar groupings occur in response to 5′ and 8′, and the

```
 1′ GDR:ES85
 4′ Full name: Treasury :Exchequer
 5′ Issue date:6/7S2
 8′ Final div date: 1/1 :5/8 81 3 1/4 0
12′  85:Exchequer  3 1/4  1981  5/8  S  6-Jul-77 OK?:Y
13′ GDR:
```

Fig. 2. Dialog sequence with grouped responses.

overall dialog is much terser. As far as the user is concerned, the advantage is not just in reducing the verbosity of the system, but also in that he comes to *think* in terms of the clustered responses so that the dialog really is mentally shortened, i.e. ES85 is the unitary *sentence* "Enter Stock 85". Such groupings of basic actions together to form single unitary sequences is a major characteristic of the formation of any skill, and it is very natural for the user to adapt his actions in this way. Note, as emphasized previously, that the change in the way the system is being used is entirely under user control—the dialog is adapting to the user's increasing skill and the system is programmed to allow such adaptation, but it is still the user who dominates the interaction by deciding when to group his inputs.

There are clearly some syntactic constraints upon responses that may be grouped in this way but they are rarely severe in practice.

C. Minimizing the mental load on the user

It is common to find systems in which much effort has been put into user aids but this effort has not been enforced and so, occasionally, the aids are

not available. Similarly one finds that dialog terminologies or command structures are similar throughout a system except for a few particular modules. In both cases the user becomes familiar with certain features of the system and then occasionally finds that what he has learnt does not always apply. This not only puts an additional memory load on the user but can also be devastating psychologically when a naive user has at last gained confidence in his familiarity with a system and then suddenly finds that his trust in himself (and system and its designer) is misplaced. This leads to the following rule:

> *Rule 9. Uniformity and consistency*: ensure that all terminology and operational procedures are uniformly available and consistently applied throughout all system activities.

This is clearly a particularly important rule for the naive user becoming familiar with the system. However, we have also found it important for skilled users *transferring* to a new system—a good standard dialog technology has the same impact as a good standard programming language—users can transfer experience from one situation to another. This has been a focal point for many designers of standardized dialog systems in recent years (Barnard *et al.*, 1981; Moran, 1981; Robertson *et al.*, 1981; Guest, 1982; Treu, 1982).

One of the most important facilities to make consistently and uniformly available is a good "help" command that enables a user to obtain selective advice from the system at any point in the dialog. There is nothing more daunting to a naive user (or any of us) than a thick user manual with masses of fine detail about the nuances of every transaction. Yet such documentation clearly should exist and there are times when particular details in it are required. In practice there is a close connection between the state of a transaction and the information that may be relevant from the user manual, and the system itself may be used to make the relevant parts of the manual available as the user requires them. This basically corresponds to the "help" facility available on many systems whereby the user can request information and advice as he goes through a transaction. The difficulty with this is to know what advice would be most valuable to the user at any point: sometimes a list of possibilities; sometimes an example; sometimes an explanation of what will result from an input, and so on. We have developed a technique that enables a simple user command structure to control a multi-level help facility:

> *Rule 10. Query-in-depth*: distribute information and tutorial material appropriately throughout the dialog system to be accessed by the user through a simple uniform mechanism. Organize the material so that the user accesses brief memory aids first at any point but has further access to more detailed expansions.

We use a question-mark, ?, as the command which is always available to the user to obtain help. It may be typed as reply at any point of a dialog and immediately elicits a short note of the main actions available at that point, or an example of data that could be entered. Typing a question mark again drives the help system to greater depth so that it gives a full range of options and their effects. Typing a further question mark may lead to further expansion or, more probably, to a manual reference. Figure 3 shows query-in-depth in action during the dialog sequence of Figs 1 and 2. The successive question marks at lines 1, 3, 10 and 27 show questioning in depth as do those at lines 38 and 40.

Sometimes the queries which the user wishes to make may involve the generation of information, perhaps through access to system utilities such as directory listing programs. In this event it is very important that the mainstream of the dialog is not interrupted and that the user is returned exactly to the point that he left with no change in the state of the environment.

The extensive use of query-in-depth may be seen as a way of embedding a *manual* interactively within the system such that is always "open" at the right page when required by the local context. This may be viewed as a means of avoiding the need to write user manuals altogether and indeed we have found that users readily transfer to new systems without using manuals. However, when a manual is written, it is certainly a rationale for linking the text back to the query-in-depth and not creating yet another level of discourse with which to confuse the user. This leads to the rule:

Rule 11. User manuals should be based on actual user dialog: illustrate the use of the system in action by showing actual dialog sequences that achieve specific objectives. Illustrate the structure of the system through dialog sequences where question marks are typed to elicit information about the system.

Thus in practice, rather than distributing a manual through the system using query-in-depth, one inserts the appropriate material and then creates a manual from an existing system by using query-in-depth to generate the text.

The availability of query-in-depth is a very great aid to on-system training and we have found that new users can most rapidly acquire knowledge of the system through actual experience of it rather than formal training. This leads to the rule:

Rule 12. Train through experience: get users interacting with an actual system as their initial training and introduce them to the system facilities such as query-in-depth.

```
GDR:?
  B (buy), S (sell), I (invest), L (list)
GDR:?
  E (enter), R (remove), L (list)
  B (buy), S (sell), I (invest), M (modify)
  D (vary dates), C (vary commission rates)
  V (vary VAT rate)
  SUM (summarize deals), SYS (system status)
  INIT (initialize GDR)
GDR:?
  E   enter a stock, client, dealer or jobber
  R   remove a stock, client, dealer or jobber
  L   list stocks, clients, dealers, jobbers
  total files, records, or free closing
  B   buy a stock
  S   sell a stock
  I   invest an amount in a stock
  M   modify an incorrect record of a deal
  D   vary date to update back records
  C   commission rate entry and listing
  V   VAT rate change
  SUM   summarize all deals for day
  SYS   give system status
  INIT   reinitialize this system as if just starting
  :   abort GDR
  \   exit to GDR: command level
GDR:?
  See Gilt Dealing Records User Manual p.12
GDR:E
Item:?
  Item to be entered: S (stock), D (dealer),
  J (jobber), C (client)
Item:S
Stock number:?
  Number of stock in range 10–99
Stock number:85
Full name: Treasury :Exchequer
Issue date:?
  e.g. 7/5/87 or 6-MAY-77
Issue date:?
  Date at which stock issued
  e.g. 30/8/64 or 30-AUG-64
Issue date:
```

Fig. 3. Examples of query-in-depth in dialog.

Many of the rules so far may be seen as techniques for avoiding the need for explicit training. Through Rule 2 we are "using the user's model" so that the activity and terminology itself is natural to the user. Through Rule 9 we are ensuring "uniformity and consistency" to aid the transfer of experience from one part of the system to another. Through Rule 15 on p. 42 we are enabling the user to make errors freely knowing he can avoid the consequences using a clean "reset" facility. *Training* is something that we have to put great effort into when we have a poorly designed system that is unnatural to the user. We can measure our success in system design by the degree to which explicit user training is unnecessary. In cases where we have installed a number of systems in succession over a period of years it has been gratifying to note that users who have learnt to operate one system have been completely happy to transfer to a new one having very different functions without explicit training, relying on the query-in-depth and reset facilities to allow them to learn about the new system through their own unguided efforts.

The dialog examples have all shown the user inputting short codes for commands rather than full English words. This is desirable to minimize typing effort since most users are not trained typists nor ever will be. However, the use of short codes increases the possibility of error in command sequences and it is important to give the user feedback as soon as possible as to what action he has actually requested. In particular, dialog sequences for different activities which go through the same initial dialog may reinforce the user's opinion that he used the right command, and the effect of finding the error later when the sequences diverge is much more devastating since it is unexpected and the user has to look back for the source of error. Hence we formulated the rule:

> *Rule 13. Make the state of the dialog observable*: give the user feedback as to the state of the dialog by making an immediate unambiguous response to any of his inputs which may cause the dialog to branch. The response should be sufficient to identify the type of activity taking place.

For example, if the prompt "Item:" could follow a command other than E, then it would be better to qualify it appropriately, e.g. "Item to be entered:". Note how this requirement interacts, however, with the previous rule 8, because the dialog of Fig. 2 bypasses the prompt "Item:". In this case the dialog programmer may either decide that it does not matter since the user is skilled (or, what amounts to the same thing at this level of discourse, is pretending to be skilled—we should not be too paternalistic towards users), or if the possibility of error at this step is very significant (e.g. a little-used command) the observability prompt may be made an output in response to the initial command:

1 GDR:E
2 Stock Entry Sequence
3 Item:

Fig. 4. Immediate feedback of state of dialog.

which will not be bypassed by a grouped initial command like ES85. However, such messages should not be used too freely or the dialog will become annoyingly pedantic—it is like the listener who repeats each sentence back at you.

Many of the prescriptions of the section can be subsumed under what Thimbleby (1980) calls *dialogue determination*, i.e. that a dialog should be neither over- nor under-determining. An under-determining dialog is one that leaves the user excessive freedom with too little information to use it, breaking Rule 13 on observability and probably Rule 10 on the capability to query for possible actions; Thimbleby notes that the user will be *mystified* by such a dialog. An over-determining dialog is one which unnaturally constrains the user, breaking at least Rule 6 and probably others requiring flexibility such as Rule 8. There are useful systemic concepts that can be seen as generative principles for several of the rules.

D. Error detection and correction

One of the significant reasons for using interactive data entry systems is to have on-line validation at source. In the light of the preceding discussion validation of data entered may be seen as part of the process of informing the user about errors in the transaction. We have found that absolute validation involving complete rejection of data is a dangerous procedure—it is rare that sooner or later some data does not come up that lies outside the prescribed "norms" and yet must legitimately be entered. It is far better to *query* suspect data, asking the user if that is really what he meant, rather than reject it. However, too frequent queries also disrupt the dialog and it is best to set norms reasonably wide so that queries are infrequent but have a final validation just before the transaction is accepted. This happens at line 12 of Fig. 1, and the clarity of the procedure is very apparent. The user knows that he is coming to a natural checkpoint where action will be taken when such validation is requested. Up to that point he is assured of being able to abort the transaction with no side effects. He also sees the overall transaction in a global fashion that was not apparent before and, because the print-out can be a standard one used for normal documents from the system, it can be a familiar form of output where errors will be most obvious. The rule is:

Rule 14. Validate data on entry by checking syntax and values, but beware of rejecting data or querying too much as being outside norms. Have the user himself revalidate major updates before acting upon them.

Errors in dialog are not essentially key problems to be avoided at all costs. Naive users should feel free to make errors as part of their exploration of the system—even highly skilled users will make errors because they will only devote to the dialog that part of their channel capacity necessary to maintain the error rate to an acceptably low level for them. In both cases the users need the *sympathetic* system noted by Cleary—they should be able to make most errors with absolute confidence that they can readily escape the consequences—or at least all but the administrative chore of going back and doing it correctly. This led us to the rule:

Rule 15. Provide a reset command that cleanly aborts the current activity back to a convenient checkpoint. The user should be able at any stage in a transaction to abort it cleanly with a system command that takes him back to a well-defined checkpoint as if the transaction had never been initiated.

We used the colon, :, as a uniformly available command to abort a transaction—it is easy for users to remember, "If you don't want to continue, type back to the computer the last character of its prompt to you, i.e. the colon". Thus one might get the sequence:

```
1 GDR:ES85
2 Full name: Treasury ::
3 GDR:
```

Fig. 5. Use of reset command.

This rule is a very important one for the user, but often the most difficult to implement for the system programmer. There may well be transactions involving many resources on a time-shared system where the "roll-back" required to allow such free aborting is almost impossible to implement. This is a very good example where making the system simple and understandable to the user may involve great complexity in the programs—however, this is clearly the right division of labour.

The user may realize that he has made errors part way through a transaction, or at the final validation and wish to go back to correct them. This is not an easy operation to offer to an unskilled user and, if such action is rarely required, it may be better to recommend that they use the abort facility of Rule 15 and go through the transaction again. Another alternative is to provide a good record update and correction facility as

discussed in the next paragraph and advise users to complete the incorrect entry and then correct it as a separate activity. We have found it useful to provide a backtrack facility that allows users to unwind the dialog in reverse order on the assumption that the error will be in the immediately preceding sequence:

Rule 16. Provide a backtrack facility that allows a user to return through the dialog sequence in reverse.

We use the "up-arrow" character as a backtrack request:

```
 1  GDR:E
 2  Item:S
 3  Stock number:85
 4  Fullname: Treasury :^
 5  Stock number:58
 6  Full name: Treasury :
 7  Issue date:6/7
 8  Type (SMLTN) :^
 9  Issue date:^
10  Full name: Treasury :^
11  Stock number:57
12  Full name: Treasury :
13  Issue date:6/7
14  Type (SMLTN):
```

Fig. 6. Correction by backtracking.

where at line 4 the user backtracks once to change the stock number entered and then at lines 8, 9 and 10 does so three times to change the number again.

The updating and correction of records is just as important as their original entry. We have learnt to assume that any field whatsoever in a record may require updating—people seem to change their name, sex and age far more frequently than one might reasonably expect. To avoid the user having to master another dialog sequence for record correction, it is possible to use exactly the same sequence as for data entry but print out field values after the prompt and take a RETURN input alone as meaning that the value has not changed. This is a particularly rapid way of modifying even a single field at a visual display terminal. The rule is:

Rule 17. Make corrections through re-entry: use the entry dialog with default field print-outs from a record as a means of correcting the record.

For example, in terms of Fig. 1, if we re-enter stock 85:

```
 1  GDR:E
 2  Item:S
 3  Stock number:85
 4  Full name: Exchequer :
 5  Issue date: 6-Jul-77 :
 6  Type (SMLTN): S :
 7  Dividends/year: 2 :
 8  Final div date : 5/8 :6/8
 9  Redemption years(s): 81 :
10  Coupon: 3 1/4 :
11  Number of calls: 0 :
12    85:Exchequer  3 1/4  1981  6/8  S  6-Jul-77 OK?:Y
13  GDR:
```

Fig. 7. Correction of record through entry sequence.

At lines 4–7 and 9–11 the user just presses the RETURN key to indicate field unchanged. At line 8 he types in a correction input. Note how the stored record in this sequence appears exactly as do the default values in Fig. 1.

IV. Extending the rules to other dialog situations

The rules described in the previous section have been targeted around the kind of formal dialog situations shown in Figs 1–7. There are many variants of man–computer dialog, however, which are still formal, or semi-formal, that go outside this framework. In this section we briefly consider some of these and the implications of the rules for them.

Chris Evans led a number of studies of computers in use to supplement human interviewing in such areas as general medicine (Card and Lucas, 1981) and experimental results suggested that the computer terminal was more acceptable than the person for many patients (Card *et al.*, 1974). The approach Evans took to making systems *user-friendly* included the use of *encouraging phrases* such as "Thank you. You are managing to answer my questions very well." Spiliotopoulos and Shackel (1981) report experiments to determine the effect of such phrases on the acceptability of computer interviews. Clearly the random emission of relevant remarks to provide a friendly halo to a conversation is a valid technique in that it corresponds to a level of human behaviour that is significant in interpersonal relations. It

goes against Rule 7 to some extent, to avoid acausality, but it may be consistent with Rule 2, in that the user's model in some contexts may include the expectation of friendly phrases with little semantic content.

The examples given in this chapter have all been based on a sequential *prompt-and-response* style of dialog which treats the interaction as a one-dimensional stream. The availability of low-cost visual displays with cursor addressing and their extension recently to high-resolution graphics has led to the developments of different forms of dialog. At the simplest level are the *menu* techniques in which the options available to a user are displayed as a table on the screen and he makes a selection from this table. In some systems this corresponds only to the table being part of a large prompt string—one is taking advantage of the speed of the display to give more information. In other systems the menu itself is accessed directly, for example by using a *mouse* to move a graphic cursor to it on Lisp Machines (Weinreb and Moon, 1981a) or by touching the screen itself on the PLATO system (Sherwood, 1977). The same rules of dialog discussed previously apply to such systems—by making more direct forms of interaction available they are highlighting the poor interface provided by the QWERTY keyboard itself for many users.

When a high resolution graphic screen is available for interaction, say with 1000 × 1000 pixels resolution, then a wide variety of new techniques become available. These have been exploited in program development systems such as that of Teitelman (1979) which allow multiple documents to appear on the screen together, some of which portray the state of ongoing processes and some of which are interactive dialog windows. Such systems are now readily implemented with the technology available (Weinreb and Moon, 1981b) but can become very complex in use. The rules given continue to apply and do not seem to require additions specifically for such systems. The new facilities do, however, allow improved implementation of some features, for example *query-in-depth* is now implementable through a separate *help* document that does not interfere with the ongoing dialog.

V. Implementing the dialog programming rules

Enforcing the dialog programming rules detailed in the previous section is best done through a standard dialog procedure which is written once and used thereafter as the only means of interacting with the user. The main programming language feature necessary for such a feature is the capability to handle variable length strings as data items and particularly as parameters to procedures. Languages such as Snobol and Lisp are ideal in the facilities they provide but even Fortran and Pascal are generally

provided with some form of primitive string data type and processing facilities. BASIC has good string handling facilities but very poor procedural modularity with its lack of parameter passing and local variables. Guest (1982) discusses the use of specialist software tools in dialog design.

The systems described in this chapter were written in the interactive systems programming language Basys II (Gaines, 1976d, 1978c; Gaines and Facey, 1977), which provides special facilities for string processing. These are based on the *compiler–compiler* (Brooker and Morris, 1962) language constructs for string analysis incorporated in a BASIC-like language that is extended to have procedure calls with parameters and local variables. The standard Basys II operating system has a dialog processing procedure embedded in it and all dialog, including that of the operating system job control language, goes through this procedure.

The procedure is called with parameters giving the prompt string, query-in-depth replies, and branches for reset and backtrack. Its operation is most readily understood by considering how each rule is implemented in turn:

Rules 1, 2 and 3 do not require any special features in the procedure. Rule 2, *Use the user's model*, implies ease of decoding strings returned by the procedure in an arbitrary format natural to the user, and this is particularly simple in Basys II. Rule 3, *Design never ceases*, implies some form of interactive programming language with a modular structure and again Basys II satisfies this requirement. However, other languages offer equally good string handling or can be provided with suitable procedures.

Rule 4. *Log activities*, is the first where the dialog procedure itself should provide facilities. One of the parameters that can be passed to it is the name of a log-file procedure. This may default either to no action or to logging the complete dialog. However, it is also possible by this mechanism to provide selective logging of the dialog at points which require study or are dependent on other parameters such as the identity of the user.

Rules 5, 6 and 7, in the context of the dialog procedure, are warnings not to make it too "clever" in its actions.

Rule 8, *Parallel-sequential trade-off*, is readily implemented. The string returned by the dialog procedure is in a global variable and the calling routine extracts what items it wants from it. When the dialog procedure is next called it checks the global variable and only if it is null does it go to the user for input. Thus the user may input several items at the same

time and they will be passed to the appropriate callers of the dialog procedure.

Rule 9, *Uniformity and consistency*, is primarily ensured by requiring all dialog to go through the standard procedure.

Rule 10, *Query-in-depth*, is implemented by passing to the procedure a table of strings which are used to reply to each successive question-mark.

Rules 11, 12, 13 and 14 depend on how the procedure is used rather than its own characteristics.

Rule 15, *Provide a reset command*, is implemented through the procedure returning not to its normal address but to the branch passed as the reset parameter. This branch can usually be set up as a default for a dialog sequence and should be the code which cleans up any actions taken during the course of the dialog before returning to the top-level program.

Rule 16, *Provide a backtrack facility*, is implemented in a similar way to Rule 15 except that the abnormal branch is to the parameter passed which will usually be the previous call on the dialog procedure or any parameter calculations immediately preceding it.

Rule 17, *Make corrections through re-entry*, is implemented by passing to the dialog procedure a computed prompt string which contains the previous value of the item requested.

Thus most of the key rules may be implemented simply and automatically through a dialog procedure. It is useful also to incorporate in the procedure facilities for diagnostic purposes and for taking the input from a command file rather than a terminal.

VI. Conclusions

This chapter has proposed an approach to *dialog engineering* which has been used in a wide range of interactive systems in the last 15 years. It has proved very successful in allowing naive users to cope rapidly and adequately with complex interactions whilst not frustrating highly skilled users. The rules proposed are intended as guidelines rather than absolutes and each is substantiated by arguments based on both theory and practice. Some aspects of the rules are specific to the particular style of dialog discussed but they are basically systemic and general to all forms of interaction. We hope that this presentation will not only act as a guide to

those attempting to create effective interactive systems but also as a stimulus to others to improve, vary, and generally develop both the rules and the entire domain of dialog engineering.

VII. Acknowledgements

We are grateful to Peter Facey, John Gedye and John Sams for their collaboration in the work reported in this chapter, and to Harold Thimbleby and Martin McGuire for access to their work.

VIII. References

Barnard, P. J., Hammond, N. V., Morton, J., Long, J. B. and Clark, I. A. (1981). Consistency and compatibility in human–computer dialogue. *International Journal of Man–Machine Studies* **15**, 1, 78–134.

Bennett, J. (1976). *Linguistic Behaviour*. Cambridge University Press, Cambridge.

Bevan, N. (1981). Is there an optimum speed for presenting text on a VDU? *International Journal of Man–Machine Studies* **14**, 1, 59–76.

Brooker, R. A. and Morris, D. (1962). A general translation program for phrase structure languages. *Journal of the Association for Computing Machinery* **9**, 1–10.

Bunderson, C. V. (1974). The design and production of learner-controlled courseware for the TICCIT system: a progress report. *International Journal of Man–Machine Studies* **6**, 4, 479–491.

Card, S. K., Moran, T. P. and Newell, A. (1980). The keystroke-level model for user performance time with interactive systems. *Communications of the ACM* **23**, 7, 396–410.

Card, W. I. and Lucas, R. W. (1981). Computer interrogation in medical practice. *International Journal of Man–Machine Studies* **14**, 1, 49–57.

Card, W. I., Nicholson, M., Crean, G. P., Watkinson, G., Evans, C. R., Wilson, J. and Russell, D. (1974). A comparison of doctor and computer interrogation of patients. *International Journal of Bio-Medical Computing* **5**, 175–187.

Checkland, P. (1981). *Systems Thinking, Systems Practice*. John Wiley, Chichester.

Cheriton, D. R. (1976). Man–machine interface design for time-sharing systems. *Proceedings of the ACM National Conference*, pp. 362–380.

Coombs, M. J. and Alty, J. L. (Eds) (1981). *Computing Skills and the User Interface*. Academic Press, London and New York.

Cox, R. B. and Martin, T. B. (1975). Speak and the machines obey. *Industrial Research*, pp. 1–6.

Danver, J. H. and Nevison, J. M. (1969). Secondary school use of the time-shared computer at Dartmouth College. *AFIPS Spring Joint Computer Conference* **34**, 681–689. AFIPS Press, New Jersey.

De Bono, E. (1979). *Future Positive*. Maurice Temple Smith, London.

Dehning, W., Essig, H. and Maass, S. (1981). The Adaption of Virtual Man–Computer Interfaces to User Requirements in Dialogs. *In* G. Goos and J. Hartmanis (Eds), *Lecture Notes in Computer Science*, p. 110. Springer-Verlag, Berlin.

Edmonds, E. (1981). Adaptive man–computer interfaces. *In* M. J. Coombs and J. L. Alty (Eds), *Computing Skills and the User Interface*, pp. 389–426. Academic Press, London and New York.

Fano, R. M. (1965). The MAC system: a progress report. *In* M. A. Sass and W. D. Wilkinson (Eds), *Computer Augmentation of Human Reasoning*, pp. 131–150. Spartan Books, Washington D.C.

Fried, B. D. (1968). On the user's point of view. *In* M. Klerer and J. Reinfelds (Eds), *Interactive Systems for Experimental Applied Mathematics*, pp. 11–21. Academic Press, New York and London.

Gaines, B. R. (1975). Analogy categories, virtual machines and structured programming. *In* G. Goos and J. Hartmanis (Eds), *GI—5. Jahrestagung. Lecture Notes in Computer Science*, pp. 691–699. Springer Verlag, Berlin.

Gaines, B. R. (1976a). On the complexity of causal models. *IEEE Transactions on Systems, Man & Cybernetics* SMC-6, **1**, 56–59.

Gaines, B. R. (1976b). Behaviour-structure transformations under uncertainty. *International Journal of Man–Machine Studies*, **8**, 3, 337–365.

Gaines, B. R. (1976c). Human factors in virtual machine hierarchies. *Proceedings Colloquium on the Influence of High Level Languages on Computer System Design*, IEE, London.

Gaines, B. R. (1976d). Interpretive kernels for microcomputer software. *Proceedings Symposium Microprocessors at Work*, pp. 56–69. Soc. Electronic & Radio Technicians, University of Sussex.

Gaines, B. R. (1977). System identification, approximation and complexity. *International Journal General Systems*, **3**, 145–174.

Gaines, B. R. (1978a). Minicomputers in business applications in the next decade. *Infotech State of Art Report on "Minis Versus Mainframes"*, pp. 51–81. Infotech International, Berkshire.

Gaines, B. R. (1978b). Programming interactive dialogue. *Pragmatic Programming and Sensible Software*, pp. 305–320. Online Conferences, Uxbridge, Middlesex.

Gaines, B. R. (1978c). A mixed-code approach to commercial microcomputer applications. *Microprocessors in Automation and Communications*. IERE Conference Proceedings No. 41, pp. 291–301.

Gaines, B. R. (1978d). Computers in World Three. *Proceedings of the International Conference on Cybernetics and Society*, Tokyo, III, IEEE 78CH-1306-0-SMC, pp. 1515–1521.

Gaines, B. R. (1978e). Man–computer communication: what next? *International Journal of Man–Machine Studies* **10**, 225–232.

Gaines, B. R. (1979). Logical foundations for database systems. *International Journal of Man–Machine Studies* **11**, 4, 481–500.

Gaines, B. R. (1980). General systems research: quo vadis? *In* B. R. Gaines (Ed.), *General Systems*, **24**, 1–9. Society for General Systems Research, Louisville, Kentucky.

Gaines, B. R. (1981). The technology of interaction—dialogue programming rules. *International Journal of Man–Machine Studies* **14**, 1, 133–150.

Gaines, B. R. and Facey, P. V. (1975). Some experience in interactive system development and application. *Proceedings IEEE* **63**, 155–169.

Gaines, B. R. and Facey, P. V. (1977). BASYS—a language for processing interaction. *Proceedings Conference Computer Systems and Technology (IERE No. 36)*, pp. 251–262. University of Sussex.

Gaines, B. R., Facey, P. V. and Sams, J. B. S. (1974). An on-line fixed-interest investment analysis and dealing system. *Proceedings European Computing Congress, EUROCOMP* **74**, 155–169.

Gaines, B. R., Facey, P. V. and Sams, J. B. S. (1976). Minicomputers in security dealing. *Computer* **9**, 9, 6–15.

Gebhardt, F. and Stellmacher, I. (1978). Design criteria for documentation retrieval languages. *Journal of the American Society for Information Science* **29**, 4, 191–199.

Gilb, T. and Weinberg, G. M. (1977). *Humanized Input*. Winthrop Publishers, Cambridge, Mass.

Goldman-Eisler, F. (1961). Hesitation and information in speech. *In* C. Cherry (Ed.), *Information Theory*, pp. 162–174. Butterworths, London.

Goodman, T. J. and Spence, R. (1981). The effect of computer response time variability on interactive graphical problem solving. *IEEE Transactions on Systems, Man and Cybernetics* SMC-11, **3**, 207–216.

Gruenberger, F. (Ed.) (1967). *The Transition to On-line Computing*. Thompson Book Co., Washington D.C.

Guedj, R. A., tenHagen, P. J. W., Hopgood, F. R. A., Tucker, H. A. and Duce, D. A. (Eds) (1980). *Methodology of Interaction*. North-Holland, Amsterdam.

Guest, S. P. (1982). The use of software tools for dialogue design. *International Journal of Man–Machine Studies* **16**, 3, 263–285.

Habermas, J. (1972). *Knowledge and Human Interests*. Heinemann, London.

Hansen, W. J. (1971). User engineering principles for interactive systems. *Proceedings of the Fall Joint Computer Conference* **39**, 523–532. AFIPS Press, New Jersey.

Harris, L. R. (1977). User oriented data base query with the robot natural language query system. *International Journal of Man–Machine Studies* **9**, 6, 697–713.

Hendrix, G. G. (1977). Lifer: a natural language interface facility. *Proceedings Second Berkeley Workshop on Distributed Data Management and Computer Networks*, TID-4500-R65, pp. 196–201.

Innocent, P. R. (1982). Towards self-adaptive interface systems. *International Journal of Man–Machine Studies* **16**, 3, 287–299.

Kelly, G. A. (1955). *The Psychology of Personal Constructs*. Norton, New York.

Kennedy, T. C. S. (1974). The design of interactive procedures for man–machine communication. *International Journal of Man–Machine Studies* **6**, 309–334.

Klerer, M. and Reinfelds, J. (Eds) (1968). *Interactive Systems for Experimental Applied Mathematics*. Academic Press, New York and London.

Krammer, G. (1980). On computer problem-solving interaction. *In* R. A. Guedj, P.

J. W. tenHagen, F. R. A. Hopgood, H. A. Tucker and D. A. Duce (Eds) (1980). *Methodology of Interaction*, pp. 279–292. North-Holland, Amsterdam.

Licklider, J. C. R. (1968). Man–computer symbiosis. *In* W. D. Orr (Ed.), *Conversational Computers*, pp. 3–5. John Wiley, New York.

Maguire, M. (1982). An evaluation of published recommendations on the design of man–computer dialogues. *International Journal of Man–Machine Studies* **16**, 3, 237–261.

Martin, J. (1973). *Design of Man–Computer Dialogues*. Prentice-Hall, New Jersey.

Mayer, R. E. and Bayman, P. (1981). Psychology of calculator languages: a framework for describing differences in user's knowledge. *Communications of the ACM* **24**, 8, 511–520.

Michotte, A. (1963). *The Perception of Causality*. Methuen, London.

Miller, R. B. (1968). Response time in man–computer conversational transactions. *Proceedings Spring Joint Computer Conference* **33**, 267–277. AFIPS Press, New Jersey.

Moran, T. P. (1981). The command language grammar: a representation for the user interface of interactive systems. *International Journal of Man–Machine Studies* **15**, 1, 3–50.

Mulhall, D. (1977). The representation of personal relationships: an automated system. *International Journal of Man–Machine Studies* **9**, 315–335.

Nickerson, R. S. (1981). Why interactive computer systems are sometimes not used by people who might benefit by them. *International Journal of Man–Machine Studies* **15**, 4, 469–483.

Orr, W. D. (Ed.) (1968). *Conversational Computers*. John Wiley, New York.

Parkhill, D. F. (Ed.) (1966). *The Challenge of the Computer Utility*. Addison-Wesley, Reading, Mass.

Patton, G. C. (1968). A message system for interactive dialog. *In* M. Klerer and J. Reinfelds (Eds), *Interactive Systems for Experimental Applied Mathematics*, pp. 271–283. Academic Press, New York and London.

Petri, C. A. (1962). *Kommunikation mit Automaten*. Dissertation an der Universitat Bonn.

Petrick, S. R. (1976). On natural language based computer systems. *IBM Journal of Research and Development* **20**, 314–325.

Pew, R. W. and Rollins, A. M. (1975). Dialog specification procedure. *Report No. 3129*. Bolt, Beranek & Newman, Cambridge, Mass.

Robertson, G., McCracken, D. and Newell, A. (1981). The ZOG approach to man–machine communication. *International Journal of Man–Machine Studies* **14**, 4, 461–488.

Sass, M. A. and Wilkinson, W. D. (Eds) (1965). *Computer Augmentation of Human Reasoning*. Spartan Books, Washington D.C.

Shackel, B. (Ed.) (1981). *Man–Computer Interaction: Human Factors Aspects of Computers and People*. Sijthoff and Noordhoff, The Netherlands.

Shaw, J. C. (1968). JOSS: experience with an experimental computing service for users at remote consoles. *In* W. D. Orr (Ed.), *Conversational Computers*, pp. 15–22. John Wiley, New York.

Shaw, M. L. G. (1980). *On Becoming a Personal Scientist*. Academic Press, London and New York.

Shaw, M. L. G. (Ed.) (1981). *Recent Advances in Personal Construct Technology*. Academic Press, London and New York.

Shaw, M. L. G. and Gaines, B. R. (1981a). The personal scientist in the community of science. *In* W. J. Reckmeyer (Ed.), *General Systems Research and Design: Precursors and Futures*, pp. 59–68. Society for General Systems Research.

Shaw, M. L. G. and Gaines, B. R. (1981b). The personal computer and the personal scientist. *In* R. D. Parslow (Ed.), *BCS'81 : Information Technology for the Eighties*, pp. 235–252. Heyden, London.

Shaw, M. L. G. and Gaines, B. R. (1982). Eliciting the real problem. *In* H. Wedde (Ed.), *International Working Conference on Model Realism*. Bonn, Federal Republic of Germany.

Sherwood, B. A. (1977). *The TUTOR Language*. Control Data Education Company, Minneapolis.

Shneiderman, B. (1980). *Software Psychology*. Winthrop, Cambridge, Mass.

Smith, H. T. and Green, T. R. (Eds) (1980). *Human Interaction with Computers*. Academic Press, London and New York.

Spiliotopoulos, V. and Shackel, B. (1981). Towards a computer interview acceptable to the naive user. *International Journal of Man–Machine Studies* **14**, 1, 77–90.

Teitelman, W. (1979). A display oriented programmer's assistant. *International Journal of Man–Maching Studies* **11**, 2, 157–187.

Thadhani, A. J. (1981). Interactive user productivity. *IBM Systems Journal* **20**, 4, 407–423.

Thimbleby, H. (1980). Dialogue determination. *International Journal of Man–Machine Studies* **13**, 3, 295–304.

Toffler, A. (1980). *The Third Wave*. Bantam, New York.

Treu, S. (1982). Uniformity in user-computer interaction languages: a compromise solution. *International Journal of Man–Machine Studies* **16**, 2, 183–210.

Turoff, M., Whitescarver, J. and Hiltz, S. R. (1978). The human machine interface in a computerized conferencing environment. *Proceedings of the IEE Conference on Interactive Systems, Man and Cybernetics*, pp. 145–157.

Walther, G. H. and O'Neil, H. F. (1974). On-line user–computer interface—the effects of interface flexibility, terminal type, and experience on performance. *Proceedings of the National Computer Conference* **43**, 379–384. AFIPS Press, New Jersey.

Wasserman, T. (1973). The design of idiot-proof interactive systems. *Proceedings of the National Computer Conference* **42**, M34–M38. AFIPS Press, New Jersey.

Weinreb, D. and Moon, D. (1981a). *Lisp Machine Manual*. Massachusetts Institute of Technology, Cambridge, Mass.

Weinreb, D. and Moon, D. (1981b). Introduction to using the window system. *Artificial Intelligence Laboratory Working Paper 210*. Massachusetts Institute of Technology, Cambridge, Mass.

Weizenbaum, J. (1967). Contextual understanding by computers. *Communications ACM* **10**, 8, 474–480.

Winograd, T. (1972). *Understanding Natural Language*. Edinburgh University Press, Edinburgh.

Woods, W., Bates, M., Brown, G., Bruce, B., Cook, C., Klovstad, J., Makhoul, J., Nash-Webber, B., Schwartz, R., Wolf, J. and Zue, V. (1976). Speech understanding systems. *BBN Report Number 3438*. Bolt, Beranek & Newman, Cambridge, Mass.

Young, R. M. (1981). The machine inside the machine: user's models of pocket calculators. *International Journal of Man–Machine Studies* **15**, 1, 51–85.