pictorial and textual information. Just what capabilities should be provided to the designer and just how the designer should work are considerations which need to be explored further with detailed empirical research. Furthermore, the question of how to train the designer is one that needs additional consideration.

This examination of needed areas of research is not complete. But it does give some flavor of some of the problems which need to be examined.

NOTES

1. Walter Perry, *The Open University* (Jossey-Bass, San Francisco, 1977).

2. Alfred Bork, *Learning with Computers* (Digital Press, Bedford, Mass., 1981), Chapter 6.

3. Barry L. Kurtz and Alfred Bork, An SADT™ Model for the Production of Computer Based Learning Material. In *Computers in Education*, R. Lewis and D. Tagg, Editors (North-Holland Publishing Co., New York, 1981).

4. Alfred Bork, *Learning with Computers* (Digital Press, Bedford, Mass., 1981), Chapter 6.

5. Description available from the Educational Technology Center, University of California, Irvine.

6. Description available from the Educational Technology Center, University of California, Irvine.

7. Thomas W. Malone, *What Makes Things Fun to Learn?* (Xerox Palo Alto Research Center, 1980).

8. Tim Teitelbaum and Thomas Reps, "The Cornell Program Synthesizer: A Syntax-Directed Programming Environment," *Communications of the ACM*, vol. 24, September 1981, pp. 563–573.

9. Alfred Bork, "Textual Taxonomy," Technical paper, Educational Technology Center, University of California, Irvine, January 1981.

The Technology of Interaction—Dialogue Programming Rules

Brian R. Gaines

Centre for Man–Computer Studies, London

1. 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,

This paper is dedicated to Chris Evans whose own work was closely related and complementary to it. His tragic death has cost this field of research one of its major contributors.

Brian R. Gaines, "The Technology of Interaction—Dialogue Programming Rules," *International Journal of Man–Machine Studies*, 14 (1981), 133–150. Copyright © 1981 Academic Press Inc. (London) Limited. Reprinted by permission of Academic Press Inc. (London) Limited and the author.

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). Now, another 15 years on, the continuing decline in computer costs and increase in reliability have made hands-on, interactive use of computers a commercial reality for end users (Gaines, 1978a) and this will become increasingly the normal mode of operation for the majority of computer users.

Despite the recognized importance of this transition to "conversational" computing, the study of the design of man–machine dialogues has only recently gained recognition as a field for the development of computer science and technology. The first book on the topic 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 is in a position to give a systematic account of what constitutes good dialogue programming. We are in the position today in *programming man–computer interaction* that we were with hardware design 30 years ago and software design 10 years ago—we are reliant on unsystematized "craft" skills passed on erratically through fortunate "apprenticeships." Through evolution many systems have become reasonably satisfactory in their basic dialogues and these may be copied in similar situations, but we have no production techniques for *dialogue engineering* on a specified, scheduled, quality-controlled, reliable basis.

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

It would be rash to pretend any work at this stage can approach a complete formulation of the techniques for "dialogue engineering." However, it does represent some further steps in the direction taken by the books noted above in beginning to systematize the rules for programming man–machine interaction through formal dialogues.

2. Programming Interaction

First let me make it clear what kinds of conversation and dialogue are under consideration in this paper. Natural language dialogue 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," followed by 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 dialogues with a wide range of "real" users.

However, there are a number of reasons why such truly "natural language" systems will probably not play major commercial roles for some years yet.

1. They still require large programs in powerful machines for their operation.
2. The typing required 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" dialogues minimizing user typing activities are essential.
3. In many applications where the computer replaces or mimics the clerk in an office the "natural" dialogue 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 dialogues with which I am concerned in this paper—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 structural and psychological requirements that carry over to the free form natural language dialogue that may one day become possible and desired. Figure 1 gives a sample of the type of dialogue under consideration—in this case a data entry sequence in a securities dealing system. The line numbers are for reference only—user input follows last colon on each line—lines 4, 7, and 8 offer default options—line 12 is validation before entry.

Some 5 years ago we wrote a paper (Gaines and Facey, 1975) attempting to analyze lessons learned in a wide range of experience of "Interactive System Development and Application" 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 the lessons learned as a set of eleven "Rules." While tightly formulated "rules of dialogue" are undoubtedly at this stage of the game still pseudoscience, it has been invaluable in further development of dialogue 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 in Gaines (1978b). 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 dialogue programming, and refine them further. To complete this section I 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.

```
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 year(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:
```

FIGURE 1: Typical dialogue sequence

There are two remarks that for me epitomize our objectives in designing interactive systems. Walter Doherty, manager of the in-house systems at IBM's Thomas J. Watson Re-

search 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 . . . 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 possible to minimize unnecessary activity on the part of the user—it has an emancipatory role (Gaines, 1978d). Cleary's remark emphasizes that the interactive interface seen by the user is actually to a "virtual machine"—too often dialogue 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 rigor 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 dialogue level—that is one objective of this paper. Cleary then goes on to say that the virtual machine should have an "understandable" and "sympathetic" face, two apparently nontechnical terms which may however be given quite definite meaning in terms of the comprehensibility of the virtual machine to the user and the availability of helpful aids to the user—much of what follows is concerned precisely with these two requirements.

3. The Basic Rules

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

Systems Analysis and Development

The first rule previously proposed is concerned with the initial design of the dialogue. In performing the systems analysis that precedes dialogue 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.

The phenomenon of assuming that what we personally know and have experienced is "obvious" is a common one for all human behavior. 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 to remember always 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.

Another rule is really a warning to the system 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 dialogue as if it were a conversation between two users mutually accepting this model.

This second rule may be seen as asking the

system analyst to listen and learn the user's vocabulary—not to impose a new language unnecessarily in an existing situation. I 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 dialogue, its vocabulary, its sequence, its consequences, appear natural to the user.

In this respect, the axiomatic, conventionalist psychology developed by George Kelly (1955) in terms of "personal constructs" has a significant role to play in formalizing the processes of systems analysis for dialogue engineering. As he notes (Kelly, 1955, p. 8): "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." And (p. 49) "A person's processes, psychologically speaking, slip into the grooves which are cut out by the mechanisms he adopts for realizing his objectives." It is the "grooves" created by the construct structure of the users that drives 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) in her *Pegasus* and *Focus* systems. These tools for the interactive elicitation of personal construct structures may now be seen as essential tools for the dialogue 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.

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, Facey, and Sams, 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 [that person], 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 ever so helpful at one moment and then forgetting the obvious the next. We have suggested (Gaines, Facey, and Sams, 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 his *adaptability*. If you as a newcomer have behavior 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 dialogue 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 dialogue *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' behavior 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 analyze these in terms of, for example, errors, broken down by user and by dialogue 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 dialogue is producing excessive quantities, it clearly needs redesign. 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. I suspect the lack of proper dialogue instrumentation is not due nearly so much to the commercial pressures commonly adduced as to the lack of appropriate dialogue "technology."

It seems probable that the logging recommended here will be subject to legislative constraints under pressure from labor organizations. Insomuch as it generates information about individuals, it would seem simplest to deal with 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 behavior of cooperative users (and if we wish to make all users cooperative ones!), the most effective technique is to use the computer in an "emancipatory cognitive" mode to portray to a user the structure and consequences of his own behavior [as done by Mulhall (1977) in his clinical studies of personal relationships]. Again one can envision that advanced dialogue processing systems will provide facilities for interactive access by users themselves to analyses of their own behavior. In human terms it is the ability to look up and ask "How am I doing?" that an effective logging system can provide.

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 dialogue, 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 toward "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 modeling of the environment. There is a general phenomenon at work there which is summed up in Rule 5.

> *Rule 5. The User Will Model the System.* Do not assume that the user is a passive static system to be controlled, modeled, 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 2, 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.

Another phenomenon which makes attempts to program adaption to the user within the system dangerous is that the user is not only coming to comprehend the system but also himself adapting to it. 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 computer or user should dominate interaction or there will be instability—if computer is to dominate it must be programmed to, and have sufficient information to, model user—if user is to dominate then computer system must be simple to understand—at present state of the art user should dominate 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 dialogue. 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 adaption to be feasible.

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 dialogue. The basis for it is one that I have developed in system theoretic terms in a number of publications (Gaines, 1976a, b, 1977): the identification of an acausal system (involving nondeterministic behavior) is very much more difficult than that of a deterministic system; people seem to presuppose 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.

All this emphasis on the system being static suggests that one must underutilize the capabilities of the computer to offer different dialogues under different circumstances, e.g., a highly tutorial dialogue to the naive user but a rapid, terse dialogue 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 Tradeoff.* 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 Figure 1 but with the user grouping his responses into clusters rather than making one at a time in sequence.

```
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:
```

FIGURE 2: Dialogue sequence with grouped responses

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 Figure 1), and goes straight on to prompt 4′. Similar groupings occur in response to 5′ and 8′, and the overall dialogue 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 dialogue 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 dialogue is adapting to the user's increasing skill and the system is programmed to allow such adaption, but it is still the user who dominates the interaction by deciding when to group his inputs.

There are clearly some syntactic con-

straints upon responses that may be grouped in this way but they are rarely severe in practice.

Minimizing the Mental Load on the User

It is common to find systems in which much effort has been put into user aids but where this has not been enforced so that every now and again the aids are not available. Similarly one finds that dialogue terminologies or command structure 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 learned 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 Rule 9:

> *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 to skilled users *transferring* to a new system—a good standard dialogue technology has the same impact as a good standard programming language—users can transfer experience from one situation to another.

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 dialogue. 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 multilevel help facility.

> *Rule 10. Query-in-Depth.* Distribute information and tutorial material appropriately throughout the dialogue 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 a reply at any point of a dialogue and immediately elicits a short note of the main actions available at that point, or an example of data that could be entered. Typing 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 dialogue sequence of Figures 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.

The extensive use of query-in-depth may be seen as a way of embedding a *manual* interactively within the system such that is al-

```
1  GDR:?
2    B (buy), S (sell), I (invest), L (list)
3  GDR:?
4    E (enter), R (remove), L (list)
5    B (buy), S (sell), I (invest), M (modify)
6    D (vary dates), C (vary commission rates)
7    V (vary VAT rate)
8    SUM (summarize deals), SYS (system status)
9    INIT (initialize GDR)
10 GDR:?
11   E enter a stock, client, dealer or jobber
12   R remove a stock, client, dealer or jobber
13   L list stocks, clients, dealers, jobbers,
14   total files, records, or free closing
15   B buy a stock
16   S sell a stock
17   I invest an amount in a stock
18   M Modify an incorrect record of a deal
19   D vary date to update back records
20   C commission rate entry and listing
21   V VAT rate change
22   SUM summarize all deals for day
23   SYS give system status
24   INIT reinitialize this system as if just starting
25   : abort GDR
26   ½ exit to GDR: command level
27 GDR:?
28   See Gilt Dealing Records User Manual p. 12
29 GDR:E
30 Item:?
31 Item to be entered: S (stock), D (dealer),
32 J (jobber), C (client)
33 Item:S
34 Stock number:?
35   Number of stock in range 10–99
36 Stock number:85
37 Full name: Treasury :Exchequer
38 Issue date:?
39   e.g. 7/5/87 or 6-MAY-77
40 Issue date:?
41   Date at which stock issued
42   e.g. 30/8/64 or 30-AUG-64
43 Issue date:
```

FIGURE 3: Examples of query-in-depth in dialogue

ways "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 Rule 11.

> *Rule 11. User Manuals Should Be Based on Actual User Dialogue.* Illustrate the use of the system in action by showing actual dialogue sequences that achieve specific objectives. Illustrate the structure of the system through dialogue 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 Rule 12.

> *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.

The dialogue examples have all shown the user inputting short codes. 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 [below] 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 learned 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 dialogue 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 what action he has actually requested. In particular, dialogue sequences for different activities which go through the same initial dialogue 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 Rule 13.

> *Rule 13. Make the State of the Dialogue Observable.* Give the user feedback as to the state of the dialogue by making an immediate unambiguous response to any of his inputs which may cause the dialogue 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 Rule 9 since the dialogue of Figure 2 bypasses the prompt, "Item:". In this case the dialogue 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 toward 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 [as illustrated in Figure 4],

```
1  GDR:E
2  Stock Entry Sequence
3  Item:
```

FIGURE 4: Immediate feedback of state of dialogue

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

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 do not come up that lie 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 [the suspect data]. However, too frequent queries also disrupt the dialogue 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 Figure 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, since the printout 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.* [Check] 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 dialogue are not essentially key problems to be avoided at all costs. The naive user should feel free to make errors as part of his exploration of the system—even highly skilled users will make errors because they will only devote to the dialogue 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 Rule 15.

> *Rule 15. Provide a Reset Command.* [There must be a 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 [shown in Figure 5]. 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 "rollback" required to allow such

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

FIGURE 5: Use of reset command

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 labor.

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 dialogue in reverse order on the assumption that the error will be in the immediately preceding sequence.

> *Rule 16. Provide a Backtrack Facility.* [There should be a facility] that allows a user to return through the dialogue sequence in reverse.

We use the "up-arrow" character as a backtrack request, where [for example], at line 4 [in Figure 6] 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 learned 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 dialogue sequence for record correction, it is possible to use exactly the same

```
1  GDR:E
2  Item:S
3  Stock number:85
4  Full name: 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):
```

FIGURE 6: Correction by backtracking

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 Reentry.* Use the entry dialogue with default field printouts from a record as a means of correcting the record.

For example, in terms of Figure 1, if we reenter stock 85 [we will get the sequence shown in Figure 7].

```
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 year(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:
```

FIGURE 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 Figure 1.

4. Implementing the Dialogue Programming Rules

Enforcing the dialogue programming rules detailed in the preceding section is best done through a standard dialogue procedure which is written once and used thereafter as the only means of interacting with the user. The systems described in this paper were written in the interactive systems programming language BASYS II (Gaines, 1976d, 1978c; Gaines and Facey, 1977) which provides special facilities for string processing. The standard BASYS II operating system has a dialogue processing procedure embedded in it and all dialogue, 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 dialogue procedure itself should provide facilities. One of the parameters that can be passed to it is the name of a logfileprocedure.

This may default either to no action or to logging the complete dialogue. However, it is also possible by this mechanism to provide selective logging of the dialogue 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 dialogue procedure are warnings not to make it too "clever" in its actions.

Rule 8, *Parallel–Sequential Tradeoff*, is readily implemented. The string returned by the dialogue procedure is in a global variable and the calling routine extracts what items it wants from it. When the dialogue procedure is next called it checks the global variable and only if it is null does it refer 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 dialogue procedure.

Rule 9, *Uniformity and Consistency*, is primarily ensured by requiring all dialogue 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 dialogue sequence and should be to code which cleans up any actions taken during the course of the dialogue 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 dialogue procedure or any parameter calculations immediately preceding it.

Rule 17, *Make Corrections Through Reentry*, is implemented by passing to the dialogue 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 dialogue 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.

5. Conclusions

This paper has proposed an approach to *dialogue engineering* which has been used in a wide range of interactive systems in the past 12 years. It has proved very successful in allowing naive users to cope rapidly and adequately with complex interactions while 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 of the rules are specific to the particular style of dialogue discussed whereas others are general to all forms of interaction. I hope that this presentation will act not only 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 dialogue engineering.

ACKNOWLEDGMENTS

I am grateful to Peter Facey, John Gedye, and John Sams for their collaboration in the work reported in this paper.

REFERENCES

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

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.

Cox, R. B., and Martin, T. B. (1975). Speak and the machines obey. *Industrial Research* (November), 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. New Jersey: AFIPS Press.

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

Gaines, B. R. (1975). Analogy categories, virtual machines and structured programming. In Goos, G., and Hartmanis, J., Eds., *GI—5. Jahrestagung, Lecture Notes in Computer Science*, 34, pp. 691–699. Berlin: Springer-Verlag.

Gaines, B. R. (1976a). On the complexity of causal models. *IEEE Transactions on Systems, Man and 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*, March, IEE, London.

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

Gaines, B. R. (1977). System identification, approximation and complexity. *International Journal of 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, U.K.

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

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). Man-computer communication: What next? *International Journal of Man-Machine Studies*, **10**, 225–232.

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)*, March, University of Sussex, pp. 251–262.

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*, May, pp. 155–169.

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

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

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

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

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

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*, May, pp. 196–201.

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

Martin, J. (1973). *Design of Man–Computer Dialogues.* Englewood Cliffs, N. J.: Prentice-Hall.

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

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

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

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

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

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

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

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

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

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

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,* December. Cambridge, Mass.: Bolt, Beranek and Newman.

A Human-Factors Style Guide for Program Design

Henry Simpson

Anacapa Sciences, Inc.

Human factors is a small but growing discipline which seeks to provide a method for taking into account human strengths and limitations during the design of computer hardware and software. In this article, I'll present a brief introduction to human factors and discuss its application to program design. I'll define six human-factors design principles and show how they can be applied to three areas of program design: data entry, display-screen design, and sequence control.

Human factors can be applied to any area in which a human being interacts with a machine. The discipline applies, or at least can and should be applied, to many aspects of man's interaction with computers. The most obvious area, and the one most people think of when considering human factors, is hard-

Henry Simpson, "A Human-Factors Style Guide for Program Design," *BYTE* (April 1982), pp. 108, 110, 114, 116, 118, 122, 124, 126, 130, 132. Copyright © 1982 Byte Publications, Inc. Reprinted by permission of Byte Publications, Inc.