Programming interactive dialogue

Professor Brian R.Gaines

Man-Machine Systems Laboratory,
Department of Electrical Engineering Science,
University of Essex,
U.K.

Abstract

The availability of low-cost interactive computing for commercial applications makes it attractive to give end-users direct access to computers in a 'conversational' mode. The need to minimize typing by the user leads to the conversational dialogues consisting of a series of 'prompts' and responses. Such dialogue sequences are now being programmed into a wide range of systems, but there are currently available only limited guidelines on appropriate programming techniques. This paper expresses the view that the programming of interactive dialogue is an important aspect of software engineering in its own right. It develops systematically a set of fifteen 'rules' for dialogue programming and discusses them in terms of user psychology. It is hoped that this conference will generate additions, extensions, and refinements, to these rules, eventually leading to generally useful guidelines for dialogue engineering.

1 INTRODUCTION

Despite the transition to low-cost 'conversational' computing in recent year (Gaines 1978) the study of the design of man-machine dialogues has been largely neglected 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: reliant on unsystematized 'craft' skills passed on erratically through fortunate 'apprenticeships'. Some systems have evolved reasonably satisfactory 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 basis.

It would be rash to pretend that this paper approaches a complete formulation of the techniques of '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 - THE BASIC RULES

This section develops an extended set of dialogue programming rules based on those in Gaines & Facey (1975). The rules are classified into 3 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.

There is one remark that for me epitomizes our objectives in programming interactive dialogue - 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"

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) 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 dialogue level - that is one objective of this paper. Clearly then notes that the virtual machine should have an "understandable" and "sympathetic" face, two non-technical 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 with these two requirements.

The type of dialogue with which we are concerned is not 'natural language' but rather the more formalized sequences of prompts and replies that are used in many commercial interactive systems. Figure 1 gives a sample of a data entry sequence in a securities dealing system.

```
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 :5/9
9       Redemption years(≤):81
10      Coupon:3 1/4
11      Number of calls: 0 :
12       85:Exchequer  3 1/4  1981  5/9  S  6-Jul-77 OK?:Y
13      GDR:
```

Figure 1 Typical Dialogue Sequence

The line numbers are for reference only - user input follows last colon on each line - lines 4, 7 & 8 offer default options - line 12 is validation before entry.

Systems Analysis and Development

The first rule 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 EXP 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.

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 CON 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.

These first two rules can 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 & Sams 1976). Rule EXP 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 CON 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 computers, that they would de-personalize tasks through their lack of personality, is very rapidly dispelled as one notes that interactive programs do have personality - usually imitating that of their designer ! It is very often the erratic human nature of the designer that comes over, being helpful at one moment and then forgetting the obvious the next. We have suggested (Gaines, Facey & 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 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. If what you do is not liked then you adjust rapidly until you gain approval, often through mimicry (which is the point of Rule CON). At the current state of the art it is clearly the system designer who has to provide the interactive system with its adaptability (see

Rules DOM and ACT for good reasons for not trying to automate this). 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 taken:

Rule INT 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 INT does really mean "never" and provision has to be made both for the designer to play a continuing, but decreasing, role in the adaption 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 LOG Log activities: use the computer to maintain selective records of system and user activities and provide programs to analyse these in terms of, e.g., 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. Unecessary 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 need 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. 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'.

User Adaption 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 it 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, etc. 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' 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. The general phenomenon at work here is summed up in the rule:

Rule MOD *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 takes place at a 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 experience and bear no direct relation to the system. Referring back to Cleary's remark quoted earlier, 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 DOM 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 type of user. As mentioned previously in rule LOG, it is possible to gather specific useful data about individuals 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 data rate about the plant 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 ACT Avoid acausality and 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 (Gaines 1976): 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 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 he 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 data is, like the Michotte illusions, 'wired-in' to the person and unavoidable.

All this emphasis on the system being static suggests that one must under-utilize 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 rules on minimizing the effort by the user in exerting such control, but we have noted one simple technique that has proved very effective in practice:

Rule PAR Parallel-sequential tradeoff: allow the user maximum flexibility to make his responses holistically (in parallel) or serially (in sequence) as he 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:6/752
8'      Final div date: 1/1 :5/8 91 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.

There are clearly some syntactic constraints upon responses that may be grouped in this way but these 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 familar 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 rule:

Rule UNI Uniformity and consistency: ensure that all terminology and operational procedures are uniformly available and consistently applied throughout the system.

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 to control a multi-level help facility:

Rule QUE 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 it again may lead to further expansion or, more probably, to a manual reference. Figure 3 shows query-in-depth in action at the start of the sequence of Figure 1.

```
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 commision rates)
7        SUM (summarize deals), SYS (system status)
8       GDR:?
9        E  enter a stock, client, dealer or Jobber
10       R  remove a stock, client, dealer or Jobber
11       L  list stocks, clients, dealers, Jobbers,
12       total files, records, or free closing
13       B buy a stock
14       S  sell a stock
15       I  invest an amount in a stock
16       M  modify an incorrect record of a deal
17       D  vary date to update back records
18       C  commission rate entry and listing
19       SUM  summarize all deals for day
20       SYS  give system status
21      GDR:?
22       See Gilt Dealing Records User Manual p.12
```

Figure 3 Examples of Query-in-depth in Dialogue

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 the rule:

Rule OBS 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 the previous rule PAR 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 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:
```

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.

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 their 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 the rule:

Rule RES 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:
```

Figure 5 Use of Reset Command

The RES 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.

One of the significant reasons for using interactive data entry systems is to have on-line validation at source. In the light of the preceeding 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 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 other documents from the system, it can be a familiar output where errors will be most obvious. The rule is:

Rule VAL 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.

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 RES 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. Some systems 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 preceeding sequence. If general transfers of control of the dialogue are to be allowed then the following rule is helpful:

Rule MIM Allow access to general facilities not specific to the local dialogue by having the user mimic the system: i.e. the user should type in both the system prompt and his reply.

For example, in terms of the sequence of figure 1:

```
1       GDR:E
2       Item:S
3       Stock number:85
4       Full name: Treasury :Exchequer
5       Issue date:6/7
6       Type (SMLTN) :Issue date:5/7
7       Type (SMLTN):
```

Figure 6 Correction by Mimicking Prompt

where at line 6 the user corrects his previous entry by typing the relevant prompt followed by the changed data. We have used this facility in many systems to allow access by

experienced users to the full Job control language (and even the programming language) during a dialogue, e.g. to check on other users, file directories, etc. Essentially control may be passed to another process without disrupting the dialogue provided, of course, the operating system supports this.

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 dialogue 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.

Rule COR Make corrections through re-entry: 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 re-enter 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:
```

Figure 7 Correction of Record Through Entry Sequence

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

3 CONCLUSIONS

This paper has formulated a range of basic rules for programming interactive dialogue. They have developed from practical experience but require validation, refinement and further extensions - I hope this conference will aid this

process. Clearly, the implementation of these rules should itself not be haphazard, and in practice we use a dialogue module which makes it easy to keep to these rules and enforces many of them.

4 ACKNOWLEDGEMENTS

I am grateful for discussions with Bruce Anderson, Ian Benest, Chris Corbett, Walter Doherty, Peter Facey, John Gedye, Tim Kennedy and John Sams.

5 REFERENCES

B.R.Gaines, Analogy categories, virtual machines and structured programming, in G.Goos & J.Hartmanis (eds.), GI - 5.Jahrestagung, Lecture Notes in Computer Science 34, 1975, Springer Verlag, Berlin, 691-699.

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

B.R.Gaines, Minicomputers in business applications in the next decade, Infotech State of Art Report on "Minis Versus Mainframes", 1978, to appear.

B.R.Gaines & P.V.Facey, Some experience in interactive system development and application, Proc. IEEE 63, June 1975, 155-169.

B.R.Gaines, P.V.Facey & J.B.S.Sams, An on-line fixed-interest investment analysis and dealing system, Proc. European Computing Congress, EUROCOMP 74, May 1974, 153-169.

B.R.Gaines, P.V.Facey & J.B.S.Sams, Minicomputers in security dealing, Computer 9(9), Sept. 1976, 6-15.

T.Gilb & G.M.Weinberg, Humanized Input, 1977, Winthrop Publishers, Cambridge, Mass., USA.

F.Goldman-Eisler, Hesitation and information in speech, in C.Cherry (ed.), Information Theory, 1961, Butterworths, London, UK, 162-174.

J.Martin, Design of Man-Computer Dialogues, 1973, Prentice-Hall, New Jersey, USA.

A.Michotte, The Perception of Causality, 1963, Methuen, London, UK.