Proceedings IEEE 63(6) June 1975, 894-911

# Some Experience in Interactive System Development and Application

(Manuscript received August 14, 1974, revised October 24, 1974.)

*Brian R. Gaines and Peter V. Facey*
*Man-Machine Systems Laboratory*
*University of Essex, Colchester, England*

**Abstract**: Minicomputers programmed in a high-level interactive language form a very attractive basis for the development of systems involving close man-computer collaboration. This paper is based on a wide range of experience of interactive minicomputer systems in commercial, medical, industrial, and scientific applications. It is first argued that the development of systems for effective man-computer collaboration requires not only interactive system use but also *interactive system development*. The designer needs to be able to tailor the system to user requirements at least partially as an experimental dialogue at a terminal with a user. There follows a critique of certain features of central computer utilities that limit their effectiveness in interactive applications, leading to a proposal for the use of minicomputer-based systems programmed, and used, interactively. We then give a number of case histories of our own experience in developing and using web systems in commercial, medical, and scientific applications From this experience, we have extracted a number of rubs for *programming interaction* between the user and computer system which we outlined. Finally, the main features of the software technology underlying these systems we briefly described.

## I. Introduction

This paper is based on a wide, but coherent, range of experience in the development and application of interactive computer systems during the past six years. It shows that practical cost-effective interactive systems can be developed simply and swiftly using minicomputers programmed in a high-level language. Two particular software engineering techniques are singled out as crucially important: that of using a *compact interpreter* (of a Basic-like language) as a standard kernel into which tailored machine-code modules for particular applications may be readily integrated; and that of *programming the human interaction* itself to achieve effective man-computer collaboration.

The applications have been diverse but have in common the requirements for low-cost systems ($10,000-$80,000) and close man-computer interaction, often involving people with no previous experience of computers. They have ranged through: commercial systems, such as securities and exchange dealing in the City of London, England [1], on-line television advertisement booking, and interactive multicenter market-research studies; medical systems, such as hospital in-patient administration [2], clinical trials administration [3], and the automation of psychological and clinical dialogues [4], [5]; control and instrumentation systems for a mass spectrometer, speech-synthesizer and human-operator studies; and a diverse range of interactive data-processing applications, e.g., in literary analysis, urban studies [6], chess playing [7], and the development of real-time computer systems under emulators [8] . They have involved the development of systems and applications software on a variety of minicomputers together with some special-purpose terminals for human interaction.

## *A. Background to Objectives*

These developments originated from disenchantment with attempts to utilize large central computers outside their role as massive scientific calculators. We were particularly concerned with applications where valuable professional knowledge and experience were being dissipated by dilution with largely clerical activities that seemed suitable for automation: for example, the administration of psychological tests, the design or styling of equipment, and marketing of products or services. A surprisingly large number of computer applications fall into this category and our later studies have included industrial and commercial automation where one can envisage the computer as taking the "clerical" load from a skilled technician or accountant.

We were puzzled as to why many past attempts to apply computers in these areas had met with major problems, particularly antipathy on the part of the users whom they were intended to serve. Our investigations of several operational systems showed up no fatal flaws in the logic of using computers, and indicated that the problems stemmed from technically minor faults which irritated users but whose effect was cumulative over a period. It was clear that such faults could exist in any system, even one involving only inter-person interaction, but they would normally be subject to continuous attrition. Somehow the self-organizing adaptive component of most human institutions was being undermined by the manner in which computer systems were being introduced.

We felt that many of these problems that arose in system development based on computers were created by the barriers between both system designer and computer, and between users and computer, barriers which did not exist when only human beings were involved. For example, one would not expect any but a small population of enthusiastic and experienced devotees to integrate effectively into any organization in which communications, already in an artificial and stilted language, had to be routed through various irrelevant and unreadable intermediate forms; replies were received after indeterminate intervals ranging from hours to days; and the majority of these belated replies indicated that messages not precisely correct in all aspects had been imperiously rejected. When these characteristics are coupled with the problems of definition involved in automating clerical or managerial functions previously carried out by people, one would expect difficulties to abound, as they have.

These barriers to effective communication between the users and the computer system were already well known, and predicated to break down with the emergence of *interactive computer utilities* [9], [10] allowing direct and immediate man-computer communication. However, it seemed to us that such rapid interactive access could be equally beneficial in breaking down the barriers between the system designer and computer, and preventing the loss of mutual adjustment and adaptation that we had noted. In particular, we placed great emphasis on the capability of implementing a trial system very rapidly and further developing it simply and swiftly as a result of user experience. If a clerk is criticized for making an error or presenting information badly, he can respond immediately and correct his mistakes or re-present the information. This immediate interaction with the source of criticism seemed a vital component in system development. A person may know that he does not want what is offered by the system and yet be unable to specify precisely an acceptable alternative. Offering several possibilities immediately while the critical reaction is still sharp is the human approach to optimization. Introducing a substantial time lag of hours or days when a computer involved undermines this process of negative feedback in system development. The system designer needed the capability of rapid and flexible communication with the computer system as much as did the user, although for different purposes.

### *B. Problems in the Use of Central Utilities*

In the early sixties, interactive access to computer systems was beginning to become available, generally via teleprinter terminals connected over modems through the telephone system to a time-sharing partition in a central EDP machine. The prime purpose of such access was interactive editing of programs intended for batch use and actual interaction with production programs was not encouraged and often not possible. On some systems, the editor had been supplemented with interactive calculation facilities based on Joss [11], [12] or Basic [13], [14] but again the prime intent of these systems was interactive programming rather than interactive programs.

We experimented with the use of the early time-sharing systems in a variety of interactive situations largely relating to clinical medicine and psychology [15], [16], but found that, despite the benefits gained from direct interactive access by both developers and users, various problems still substantially detracted from the value of the systems.

1) *Reliability of Access*: The importance of remote interactive users finding the system available when they try to access it cannot be over-emphasized. Many applications—the administration of hospital-bed availability, for example— depend on continuous 24-h service. Even when a delay in access can be tolerated theoretically, however, it can be psychologically devastating to the remote user community.

A remote user of the time-sharing service is isolated from the staff controlling it and becomes totally so when the system is not available. In particular, he is not aware when the system has become available again and is faced with psychologically disturbing uncertainty as to whether to try again and face further rejection or wait until the system may possibly have become available.

It is a natural human reaction to attempt to avoid such a conflict situation, and it can be avoided by ceasing to use the computer-based system. Since most of our potential users had no particular enthusiasm for computers, often quite the contrary, any demotivating influence was a major problem.

We found that time-sharing systems run as an adjunct to primarily batch-processing utilities did not have a sufficiently high level of accessibility since this was not relevant to their main function. The user of a batch-processing system does not notice, for example, the 2--h scheduled downtime for maintenance or other purposes. The efficient and cost-effective organization of a central batch-processing utility has a variety of solutions, many of which are not compatible with continuous interactive access. Only when the remote user is given first priority and the system is administered primarily with his requirements in mind do these problems not arise.

2) *Cost of Access*: The cost of accessing a remote timeshared system tends to be high because of the following reasons.

a) Giving priority to the interactive user does not lead to cost-effective use of the central utility.

b) The very existence of a job connected to a terminal, even when no computation is under way, ties up expensive resources in core buffers, peripheral drivers, and line equipment.

c) The costs of communication are themselves high. In Britain a daytime call over the public network more than 50 mi away can cost some $20/h. This makes nonsense of the decreasing costs of access to the computer itself. This problem was further exacerbated by the difficulties faced by our users in dialing up over the public network. It was desirable that the terminal be available at all times, and having to dial up and log in before access was again not conducive to use of the system.

3) *Uncontrolled Response Time*: It is well known that the response-time characteristics of a time-sharing system lead to excessively long waits when the system is overloaded. This will clearly happen to any system and is not peculiar to a central utility. What is less apparent, however, is that overloading is a function of the user population and that it, and hence the response time, can be controlled far more readily as that population becomes homogenous and well defined. The active users of a central utility fluctuate wildly and uncontrollably in their characteristics and the load they place upon it. Clever scheduling techniques can reduce the effects of this fluctuation and give priority to certain classes of user, but they rarely remove it completely.

We found, in practice, that the normal centralized system could not give a well-defined maximum response time to even the virtually zero-computation interactive user, and that again the sheer variability of system response time without apparent cause was demoralizing to the remote user. In human conversation, hesitation plays an important part in indicating significance [17] and our later experience has indicated that the relation between time taken to respond and activity requested is recognized naturally by users.

4) *Inflexibility in Communication Protocols*: Most central utilities are naturally matched to some form of typewriter, or equivalent visual display, as a remote terminal. We found it necessary, for example, in psychological testing, to use random-access microfilm, or audio terminals which, although operating within the same character set as a typewriter terminal, did not necessarily obey the same communications protocol. For example, one commercial remote microfilm terminal formats its messages with STX and ETX which is quite correct by international standards but not expected from a typewriter terminal. It so happens that ETX (control C) is the return-to-monitor control command in some executives which made it impossible to use the terminal without modification. Similar problems were encountered with the use of RUBOUT on ASCII terminals as a significant character (meaning erase last character) by many executives. This character is frequently generated by a noise on the line to a remote terminal and users find it exasperating to see the last character of their message deleted while they are thinking about the next (if they understand what is happening!).

These problems can clearly be overcome by redesigning the communication protocols. Again this proved so difficult as to be impossible in using a general central utility. Only. when the machine was dedicated to our users were we able to overcome the problem.

### *C. Overall Objectives*

None of the problems noted in the previous section make it impossible for a central computer utility to form the basis of practical cost-effective interactive systems. However, they are significant limitations on both cost, and human, factors in the applicability of such systems, and they led us to investigate the possibility of developing interactive timeshared systems based on minicomputers which could be dedicated to a small group of users. With our emphasis on simple swift interactive program development, a high-level problem-orientated language and multiuser operating system seemed essential. In the early sixties, the gulf between the simple laboratory minicomputer and the large EDP machine was far greater than it is today, particularly in availability of such software. However, the development of operating systems such as Thor for the PDP1 [18] and TSS8 for the PDP8 [19], [20], and of languages such as joss on the 4096-word Johnniac [11], [12] or Telcomp on the PDP7 [21], indicated that suitable facilities could be made available.

Since we were particularly concerned with the problem of effective seminatural language interaction with users, often for nonnumeric purposes, the main limitation of the existent minicomputer calculator languages was inadequate facilities for the general treatment of character strings. Weizenbaum's Eliza [22], [23] set us a standard for the type of string-processing necessary to handle natural conversational interaction. We also needed numeric computation and the ability to control and access large data structures on files. Clearly Cobol or PL1 could have coped with the problem and Snobol [24] was nearly ideal in the facilities it offered, particularly for the contextual analysis of strings [25]. However, these were large-machine languages and it was an open question as to whether comparable facilities could be offered in an interactive language on a minicomputer. We felt that given the flexibility to tailor a language logically to both applications and implementation, a minicomputer-based high-level language interactive system could be developed which would rival very much larger hardware/ language systems.

Thus we ended our preliminary studies of the use of interactive computers, largely in psychology and medicine, with the conviction that minicomputer-based systems programmed in a high-level interactive language could provide a highly effective basis for the development of practical and cost-effective systems designed for close man-computer collaboration. Our first experience in implementing this type of system was, probably very fortunately, on a machine which made available very small store partitions. The TSS8 [19], [20] was a timeshared version of the PDP8 with a very advanced monitor for its time, but offering each user only a 4096 × 12 bit word core partition plus access to file space on a fixed-head disk. We added to our more philosophical system design requirements the rather pragmatic one of fitting the complete language system, both run-time and program editing, together with a reasonable sized user program, into 4K words of store. This did not prove difficult in practice and it has been a major virtue of all later generations of the system on other machines that they are usable with very small configurations, making interactive high-level language programming feasible in truly mini systems.

In summary, our initial objective was to design a minicomputer-based interactive system to provide a basic module for system development in applications involving close man-computer collaboration that included the following features.

1) *Cost Effective*: This should stem in hardware terms, from the use of minicomputers available as a collection of hardware modules that could be minimally configured to the requirements of a particular application, and, in software terms, from the ease and speed of system development in an interactive high-level language.

2) *Reliable*: Again, small minicomputer configurations designed for industrial use with minimal electromechanical peripherals should be capable of maintaining 24-h day continuous service over long periods.

3) *Responsive*: We aimed to tune the system to the interactive user with short-time quanta (typically 100-200 ms) and a priority of input-output over computation.

4) *Flexible in Terminal Support*: We aimed to make input-output as directly accessible as possible, consistent with system security, and, in our own monitors, to parametrize all terminal protocols to give high generality.

5) *Supported Natural-Language-Like Interaction*: We aimed for no constraints on the syntax or semantics of interaction, and a language which made it simple to program any interactive protocol natural to the user.

6) *Supported Interactive System Development*: We wanted to be able to modify at one terminal the system being used at another terminal, a dangerous possibility but expressing the ultimate degree of interactive system design.

7) *Unrestricted in Its Data-Processing Capabilities*: We wanted unrestricted access to the data-processing facilities of the configuration, i.e., not a specialist language geared only to string processing. A minicomputer has a significant "number-crunching" capability that may be used advantageously, e.g., in batch-mode overnight.

8) *Modular and Flexible*: We saw ourselves as setting up a skeleton system that could be rapidly adapted for any particular purpose.

### *D. Organization of Paper*

Our early experience with interactive systems and the objectives outlined in the previous section led us to develop a compact interactive language, Basys [26], whose syntax was closely modelled on Dartmouth Basic but whose facilities were closer to those of a systems programming language. The main features of Basys are the compactness of the complete language system (34K bit of program and tables + typically 24K bit per user program area); the aids to rapid interactive program development; the facilities for handling data structures and records with mixed lengths and types of operand; the facilities for the manipulation and contextual analysis of arbitrary-length 7-bit character strings; and the ease of extension of the language to integrate in new commands providing special-purpose facilities. There are now many versions of the language in use under the names Quasic [27], Quasac [28] (both for TSS8), Aims [29] (PDP11), Minsys [30] (Minic I), and Basys [31] (PDP9, PDP10, and PDP11), and a detailed technical presentation of its facilities is given in [26]

It is not our intention in this paper to attempt to propagate Basys as a programming language. A final section on the language itself and its implementation extracts only those aspects of it that have been particularly important to the success of applications and which have more general implications. In the next section we introduce a sample of the applications of a number of Basys systems to illustrate the immediate practical potential of minicomputer-based systems programmed in a high-level interactive language. Out of these application studies, we have developed some basic precepts for programming the interactive dialogue between the user and the computer and these are reported in a further main section.

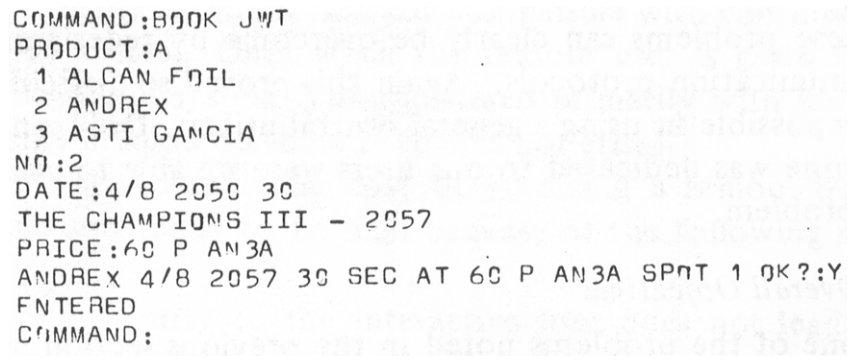
## II. SOME APPLICATION STUDIES

No matter how cogent and convincing the arguments one can develop for the application of certain systems engineering techniques, their most important attributes will always remain their utility in actual system development. There are so many interacting variables in reality that neither success nor failure of a project can logically validate, or invalidate, a specific technique. However, the credibility of the arguments is greatly affected by the situations in which they have been generated. The following sections present brief reviews of some representative Basys systems with special emphasis on those aspects relating to man-computer collaboration and speed of development.

### *A. Commercial Applications*

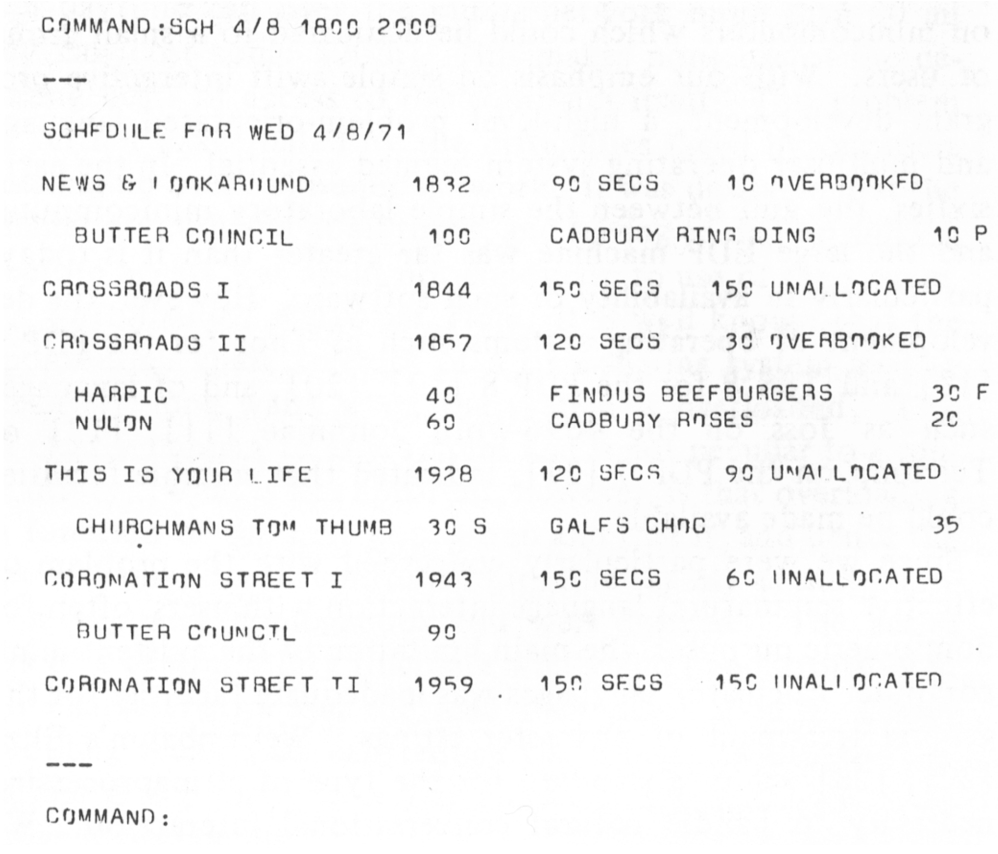
Although Basys was not designed with business systems in mind, we have regarded commercial applications as the ultimate test of system utility, particularly cost effectiveness. When the first TSS8-based system became available, we found immediate interest in its potential for business use. A typical class of application was that which requires the control of a limited nonhomogeneous resource to be allocated to a variety of clients by a number of agents, e.g., the sale of package holidays, and the placement of secretarial staff. A good example is a system developed for the sale of television advertising time. The task is simple in concept but involves continually updating a complex data structure, searching it for available slots according to various criteria, and detecting and resolving conflicts such as competitive products in adjacent slots.

The time-scale for the development of an advertisement booking system on the TSS8 is of particular interest because it illustrates the capability to generate trial systems rapidly. A model system with a range of facilities for inserting or deleting slots, booking slots, searching for available slots, examining the scheduled advertisements, etc., was developed in 20 man-hours over one weekend on a remote terminal to the TSS8. The system was highly interactive and designed for users completely new to computers (the conventional system for booking is an elaborate array of roller blinds with inserts for booking cards). We have used it to exemplify many of the techniques for programming interaction discussed in the next main section. (Later, Fig. 12 will show a dialogue with the system in which the initial query prompts a list of possible commands and then the booking sequence is entered. Figs. 13 and 14 will show how bookings are made and modified.) All inputs are validated on entry, e.g., in Fig. 1, the agency is found uniquely but the product is not, which causes all products for that agency beginning with the input string to be listed for further selection (if there were none, those containing the string would be listed; if none again, all products for the agency would be listed). At the termination of the entry sequence, the proposed action is printed for confirmation. Fig. 2 shows a request for the schedule between 18.00 and 20.00 on August 4, which is printed out formatted as a clear and simple document.



```
COMMAND:BOOK JWT
PRODUCT:A
 1 ALCAN FOIL
 2 ANDREX
 3 ASTI GANCIA
NO:2
DATE:4/8 2050 30
THE CHAMPIONS III - 2057
PRICE:60 P AN3A
ANDREX 4/8 2057 30 SEC AT 60 P AN3A SPOT 1 OK?:Y
FNTERED
COMMAND:
```

**Fig. 1: Resolution of ambiguity in a TV booking sequence. The user is booking a slot for Andres on behalf of agency JWT. He gives just the first letter of the product which is ambiguous-the system lists all those beginning with that letter to resolve the ambiguity.**



```
COMMAND:SCH 4/8 1800 2000
---

SCHFDULE FOR WED 4/8/71

NEWS & LOOKAROUND         1832      90 SECS     10 OVERBOOKFD

  BUTTER COUNCIL          100      CADBURY RING DING        10 P

CROSSROADS I              1844     150 SECS   150 UNALLOCATED

CROSSROADS II             1857     120 SECS    30 OVERBOOKED

  HARPIC                   40      FINDUS BEEFBURGERS       30 F
  NULON                    60      CADBURY ROSES            20

THIS IS YOUR LIFE         1928     120 SECS    90 UNALLOCATED

  CHURCHMANS TOM THUMB    30 S     GALFS CHOC               35

CORONATION STREET I       1943     150 SECS    60 UNALLOCATED

  BUTTER COUNCTL           90

CORONATION STREFT TI      1959     150 SECS   150 UNALLOCATED


---

COMMAND:
```

**Fig. 2. Document production—a TV schedule printout.**

This type of application seems a very natural one for an interactive minicomputer in Basys. It is essentially control of a medium-size data base with interactive, prompted, and validated data entry, and production of a variety of documents. The numerical data-processing and file-sorting loads are low although not insignificant. There have been a number of successful business systems using Basys in this type of application, notably in London for foreign exchange and gilt-edged security dealing. This last application illustrates the use of more complex numerical data processing and graphic terminals and is outlined in the next section.

### *B. A Commercial Application—Gilt-Edged Security Dealing*

Between August and December 1972, we developed a display-based interactive minicomputer system programmed in Basys for gilt-edged security broking that was installed in the London Stock Exchange in January 1973 [1]. The configuration is illustrated in Fig. 3. It consists of 20K × 16 bit core PDP11/20, 256K × 16 bit fixed-head disk, two 145K × 16 bit magnetic tape units, 6 channels of 32 line × 48 character television displays, 5 storage-tube graphic displays, and a 30CPS teleprinter. It is installed in the same office as the twelve dealers it serves and is continuously updated with information on price changes over a radio link from the exchange floor. For each security, current yields and deviations from the general trend are calculated and displayed on television monitors (Fig. 3). The graphic terminals enable dealers to request individual stock-by-stock comparisons, fitted trend lines, and other facilities for investment analysis.

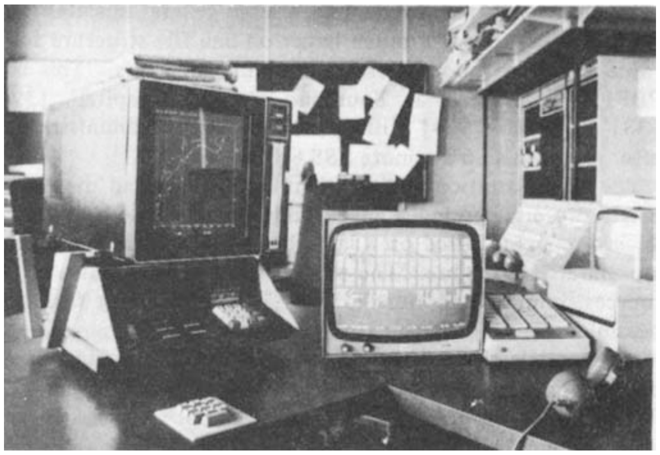


**Fig. 3. Gilt-edged security dealing station in Stock Exchange office. Foreground left: graphic display and keyboards. Center: television monitor display of prices and yields. Right: telephone key system. In the right background can be seen the computer.**

The physical dealing situation before a computer was installed was that twelve dealers and ancillary staff, each with a multiline key-and-lamp telephone desk set, were in a small office grouped in clusters specializing in different types of stock. Price information was telephoned in from the exchange floor and written up on a rollerblind display in a maximally visible position. A list of prices and yields of all stocks, calculated from the previous night's closing position, was circulated each day. Graphs of some bases for stock comparison, such as redemption yield as a function of period to redemption, were drawn by hand. Desk calculators were available for the recalculation of yields as prices changed during the day. A timesharing computer bureau was used once a day to give stock by stock comparisons based on the last three months' prices.

As far as the dealers are concerned, the computer system has three distinct functions.

1) *Immediate Market State:* Each dealer has a television monitor (middle Fig. 3) and keyboard enabling him to select any one of 6 pages (32 lines of 48 characters) of information relating to stocks (these are driven by a central MOS store updated a line at a time by the computer). The pages are divided into 3 groups of two—the groups being short/medium/long Redeemable stocks—the first page of each pair contains the latest price/touch and yield information on each stock—the second page of each pair contains details of the most recent price changes for the

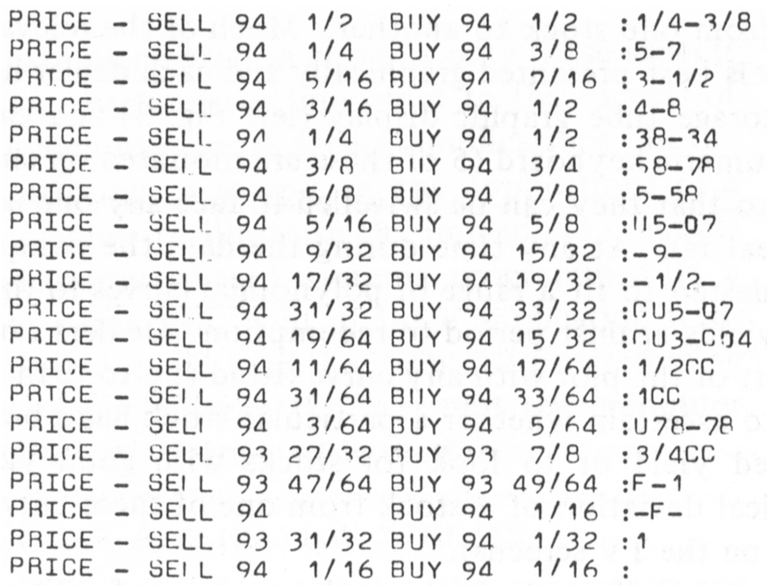
corresponding stocks. The information is updated continuously by an operator in radio communication with the exchange floor working at a 30-CPS typewriter. The computer calculates the yields from the prices and internally stored information to produce a continuously updated display—changes within the last few minutes are asterisked and the time of most recent change is recorded on the appropriate pages. News flashes are also put up on one page and relevant financial information (spot and forward sterling, etc.) on another.

2) *Stock Comparison:* Many major decisions revolve around which particular stock to sell or buy and whether to switch funds from one stock to another. Much of the relevant information is best presented graphically and each dealer has access to a storage tube graphic display (left Fig. 3) and calculator-style numeric keyboard (5 of these are mounted on small turntables so that they can be swivelled to face any one in a group of 4 dealers). At any time during the day, the computer can be requested to fit a range of polynomial curves to the plot of stock yields against period to redemption. Dealers can display any part of the plot with any curve fitted (up to fifth order) in order to ascertain whether a particular stock has a better than expected yield or to look for stocks with good yields (the numerical deviation of a stock from one of these curves is also shown on the TV screens).

The graphic displays also have alphanumeric facilities and are used to give a dealer individually requested stock by stock comparisons. All the current information on any two stocks may be displayed in conjunction for comparison. Additionally, the system maintains a record of the last 64-days characteristics of a stock (maximum and minimum price and yield, etc.) and the comparison includes the maximum and minimum of various differential parameters of the stocks (price ratios, yield differences, etc.) over a period for comparison with current values.

3 *Investment Research:* One function of a dealer is to act as an investment analyst investigating the relative merits of various stocks, coming to some understanding of the reasons for the past behavior of stocks, both absolutely and relative to one another, and trying to predict what future trends will be. This is because much of the time future trends are a function of extra-market influences and attention is focused on events in the politico-economic world. There are also many occasions, however, when the market is comparatively quiet, possibly seeking to attain some new equilibrium after events have modified the basis of financial logic. At such times, it is useful to be able to "browse" through stocks, comparing their merits on different bases, and generally looking for individual stocks whose rating is inappropriate in relation to the market as a whole. Switching in or out of such stocks may be attractive to particular clients even when they are not otherwise inclined to be active in dealing. The system gives such "browsing" facilities at the graphics terminals—e.g., plots may be made of the range of price ratios of one stock to another over the last three months, showing the effect of different tax liabilities, centered on the mean value to indicate deviations, and with a marker for the current position. The stock-by-stock comparisons already mentioned may also be used for such investment analysis, and there is a facility for an overnight printout of a cross-comparison of all stocks against all others with markers for parameters which are within 10 percent of their peak value during the last three months.

Thus the dealer has, literally at his fingertips, complete up-to-date figures of recent price changes, the current market prices, the associated yields, and deviations from the fitted curve of yields for every stock. He is able to quote to clients, in whatever detail is required, the exact situation for every stock. In addition, he can request at the graphics terminal a cross-comparison of two stocks that he is discussing with a client and, within a few seconds, be able to quote comparative figures on the relative merits of the two stocks and their behavior over the past three months. On a longer term basis, the dealer may browse through a wealth of data about stocks for purposes of investment analysis, and to confirm or invalidate his appraisals of stocks based on other sources of information.



```
PRICE - SELL 94  1/2  BUY 94  1/2  :1/4-3/8
PRICE - SELL 94  1/4  BUY 94  3/8  :5-7
PRICE - SELL 94  5/16 BUY 94  7/16 :3-1/2
PRICE - SELL 94  3/16 BUY 94  1/2  :4-8
PRICE - SELL 94  1/4  BUY 94  1/2  :38-34
PRICE - SELL 94  3/8  BUY 94  3/4  :58-78
PRICE - SELL 94  5/8  BUY 94  7/8  :5-58
PRICE - SELL 94  5/16 BUY 94  5/8  :U5-O7
PRICE - SELL 94  9/32 BUY 94 15/32 :-9-
PRICE - SELL 94 17/32 BUY 94 19/32 :-1/2-
PRICE - SELL 94 31/32 BUY 94 33/32 :CU5-O7
PRICE - SELL 94 19/64 BUY 94 15/32 :CU3-CO4
PRICE - SELL 94 11/64 BUY 94 17/64 :1/2CC
PRICE - SELL 94 31/64 BUY 94 33/64 :1CC
PRICE - SELL 94  3/64 BUY 94  5/64 :U78-78
PRICE - SELL 93 27/32 BUY 93  7/8  :3/4CC
PRICE - SELL 93 47/64 BUY 93 49/64 :F-1
PRICE - SELL 94       BUY 94  1/16 :-F-
PRICE - SELL 93 31/32 BUY 94  1/32 :1
PRICE - SELL 94  1/16 BUY 94  1/16 :
```

**Fig. 4. The “shorts language.” The price printout on the left shows . how the program interprets the coded price range information previously entered on the right. The initial price already stored is 94½.**

The application program suite generally consisted of some 40 Basys programs, on average about 100 lines long, written to a large extent as independent entities, each working on one single central well-defined file structure. Many of these were developed and debugged on the system while it was in use and much of the fine detail of output formats, etc., was essentially an interactive dialogue between users and programmer while the system was in operation. In this type of application, where the role of the computer is novel for all concerned, while a functional specification is vital in advance, a great deal of the design of proper user-computer interaction can only be done at the stage where the possibilities can be demonstrated and experienced in reality. The facility of interactive system modification allowed the final stage of system design to be a snuggling together of computer and user into a reasonably symbiotic relationship—not one where the computer was highly parasitic on the users’ powers of adaptation!

A good example of the use of Basys string processing to adapt the system to the user is the “shorts language” developed for input and display or prices of the short stocks. These prices are communicated in a dealer’s jargon which has various short forms for common situations and, on analysis, turns out to be consistent, unambiguous, and less prone to errors in verbal transmission than number strings. Fig. 4 shows the language in use: only the buying and selling fractions are communicated (the number of whole pounds, the “big figure,” is obvious with short stocks); sixteenths are referred to by the numerator only—5 is 5/16; “under” means less by 1/32, “close under” means less by 1/64—these were written, e.g., U3 (=5/32), CU3 (=11/64); similarly for “over” and “close over;” “either side” means a price range from -1/32 to +1/32 about a center price—this was written e.g., -7- (=13/32 to 15/32); “close to close” meant ±1/64 e.g., 1/2CC (=31/64 to 33/64). The highly favorable dealer reaction to the changeover from fractions to the commonly used jargon stressed once again the importance of using in full the computers encoding/decoding capabilities to match user expectations and requirements.

### *C. A Health Services Application—Hospital Administration*

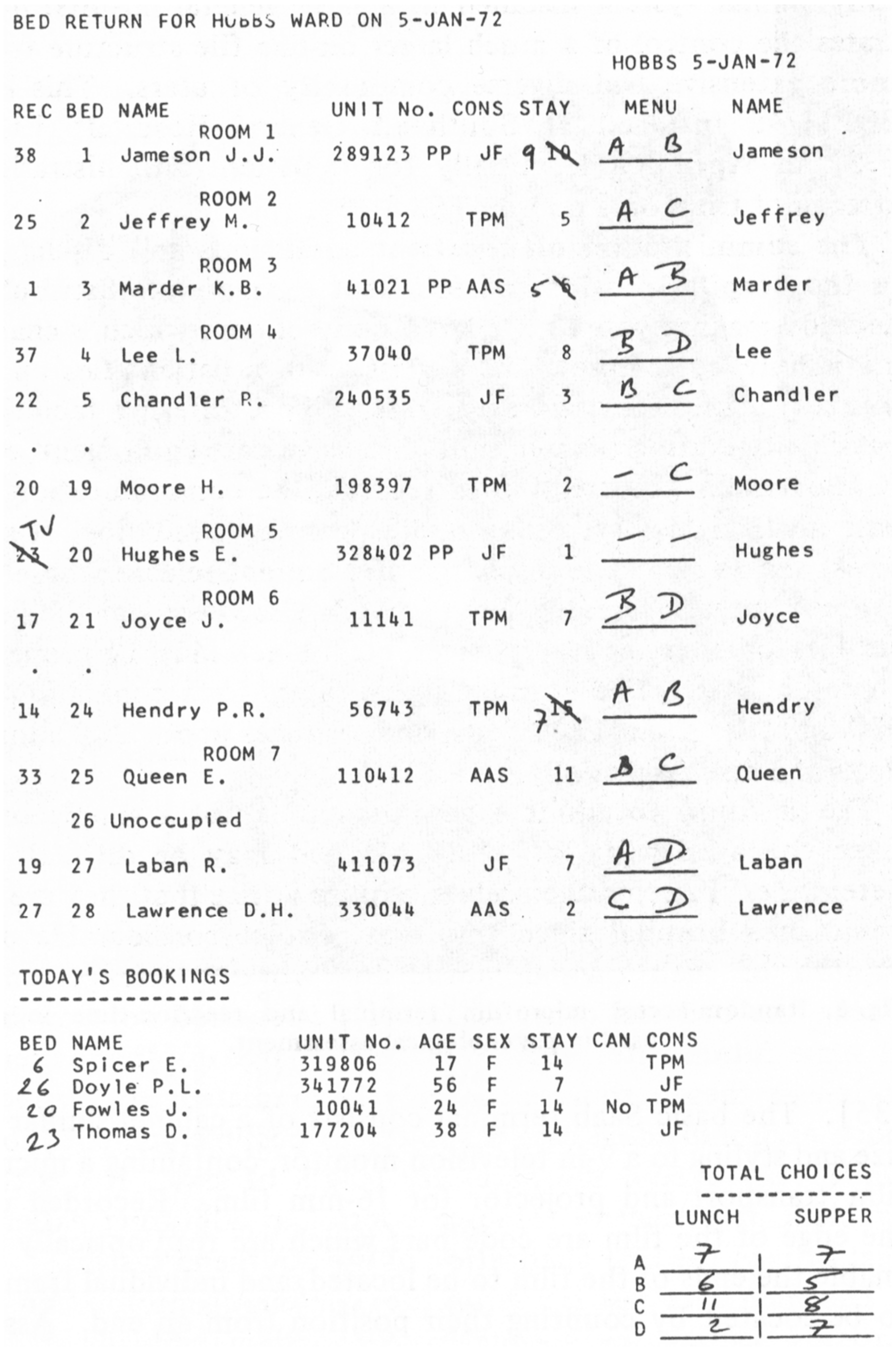
The securities system described in the previous section typifies the larger type of application where the use of an interactive minicomputer programmed in a high-level language can lead to both better man-computer collaboration and a short system development period. In that application, the data-base is comparatively small, as is the number of users. A technically similar system installed in a large general hospital illustrates the control of a much larger on-line file structure for a more extensive and diverse community of users. This is a PDP11/45 installed at Southend General Hospital (supported by the Department of Health and Social Security as part of its experimental program) [32], [33] in April 1974, initially for in-patient administration, after pilot trials on a remote TSS 8 [2].

The administration of in-patient admissions and discharges in the hospital provides an excellent example of distributed decision-making based on a large body of data which is changing minute by minute. The admission of a patient ties up resources, clinical and nursing staff, a bed, catering facilities, possible operating theater time, intensive care equipment, etc. It also entails the creation of records, the communication of past medical history, calls on the pharmacy, radiological services, and so on. The discharge of a patient releases these resources but also creates new demands elsewhere—on the general practitioner, welfare services, etc., which must be properly communicated. The hospital group caters for a population of 300 000, has some 1200 beds, and averages about 200 admissions and discharges daily.

The decision to admit a new patient is not a simple one. Even the availability of a suitable bed may be difficult to determine. Patients themselves require notice that they are to come into hospital since this may involve considerable disruption of their own domestic and working arrangements. Hence the bed and other resources are required not now but at some future date. However, the discharge of other patients to release these resources can only be predicted, not planned, since each individual case is subject to possible complications which may extend the necessary length of stay. It was this problem of predictive resource control in admitting patients that formed the first application of the Southend system.

The main data base is a set of files containing information on patients waiting for hospital places, patients actually in the hospital, and those previously discharged. These are created and updated by clerks through interactive dialogues at teleprinters or visual displays as in the other applications, and are used to generate a wide range of displays and documents tailored to the requirements of particular hospital staff. The mechanism by which the state of the active data base is continually validated, and the predictive information is generated, is particularly interesting since it illustrates the role of document production for information acquisition as an alternative to terminal interaction.

Prior to the installation of the computer system, a catering return used to be circulated to the wards each day to determine the meals required. Now this document is produced by the system as shown in Fig. 5 and contains an additional column giving the estimate currently held of each patient’s expected length of stay. The nursing staff which completes the catering return also adjusts the length of stay information if necessary and these changes are entered centrally into the system. Validation is inherent in this approach because the nurse is also aware if the patient has been discharged or moved to another ward, and corrects the return if this information has not been entered into the data-base. Thus the nurses have a natural channel of communication with the computer that is a simple extension of their current activity (and an attractive one because they can see what the central administration thinks their workload is) and does not involve direct interaction with terminals.

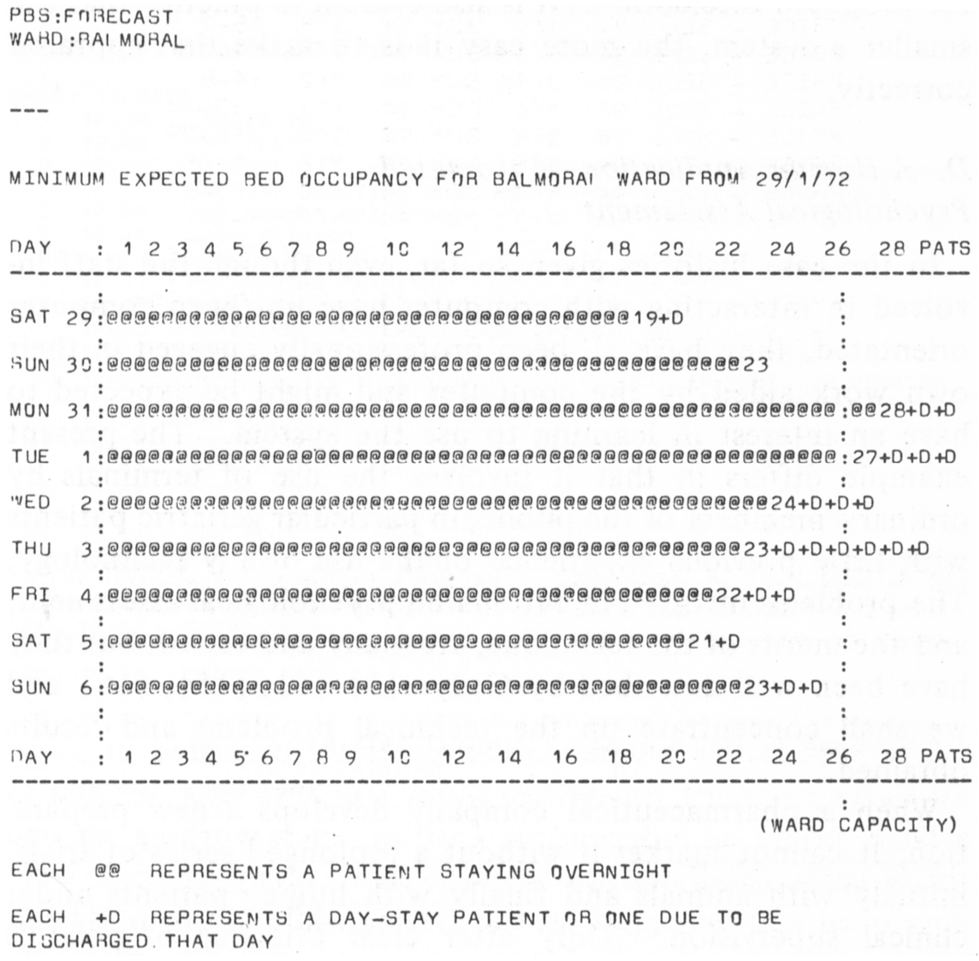


```
BED RETURN FOR HUbbS WARD ON 5-JAN-72

                                               HOBBS 5-JAN-72

REC BED NAME                UNIT No. CONS STAY     MENU      NAME
            ROOM 1
38   1  Jameson J.J.        289123 PP  JF 9 ~~10~~  A  B     Jameson

            ROOM 2
25   2  Jeffrey M.           10412    TPM    5     A  C      Jeffrey

            ROOM 3
 1   3  Marder K.B.          41021 PP AAS 5 ~~6~~   A  B     Marder

            ROOM 4
37   4  Lee L.               37040    TPM    8     B  D      Lee
22   5  Chandler R.         240535     JF    3     B  C      Chandler

 .   .

20  19  Moore H.            198397    TPM    2     -  C      Moore
TV
~~23~~  20  Hughes E.       328402 PP  JF    1     -  -      Hughes
            ROOM 6
17  21  Joyce J.             11141    TPM    7     B  D      Joyce

 .   .

14  24  Hendry P.R.          56743    TPM 7 ~~15~~  A  B     Hendry

            ROOM 7
33  25  Queen E.            110412    AAS   11     B  C      Queen

    26 Unoccupied

19  27  Laban R.            411073     JF    7     A  D      Laban

27  28  Lawrence D.H.       330044    AAS    2     C  D      Lawrence

TODAY'S BOOKINGS
----------------

BED NAME              UNIT No. AGE SEX STAY CAN CONS
 6  Spicer E.          319806   17  F   14       TPM
26  Doyle P.L.         341772   56  F    7        JF
20  Fowles J.           10041   24  F   14   No  TPM
23  Thomas D.          177204   38  F   14        JF

                                            TOTAL CHOICES
                                            -------------
                                           LUNCH   SUPPER

                                       A     7   |   7
                                       B     6   |   5
                                       C    11   |   8
                                       D     2   |   7
```

**Fig. 5. Catering return circulated to wards to ascertain meals required and also used to obtain updated estimates of expected length of stay.**

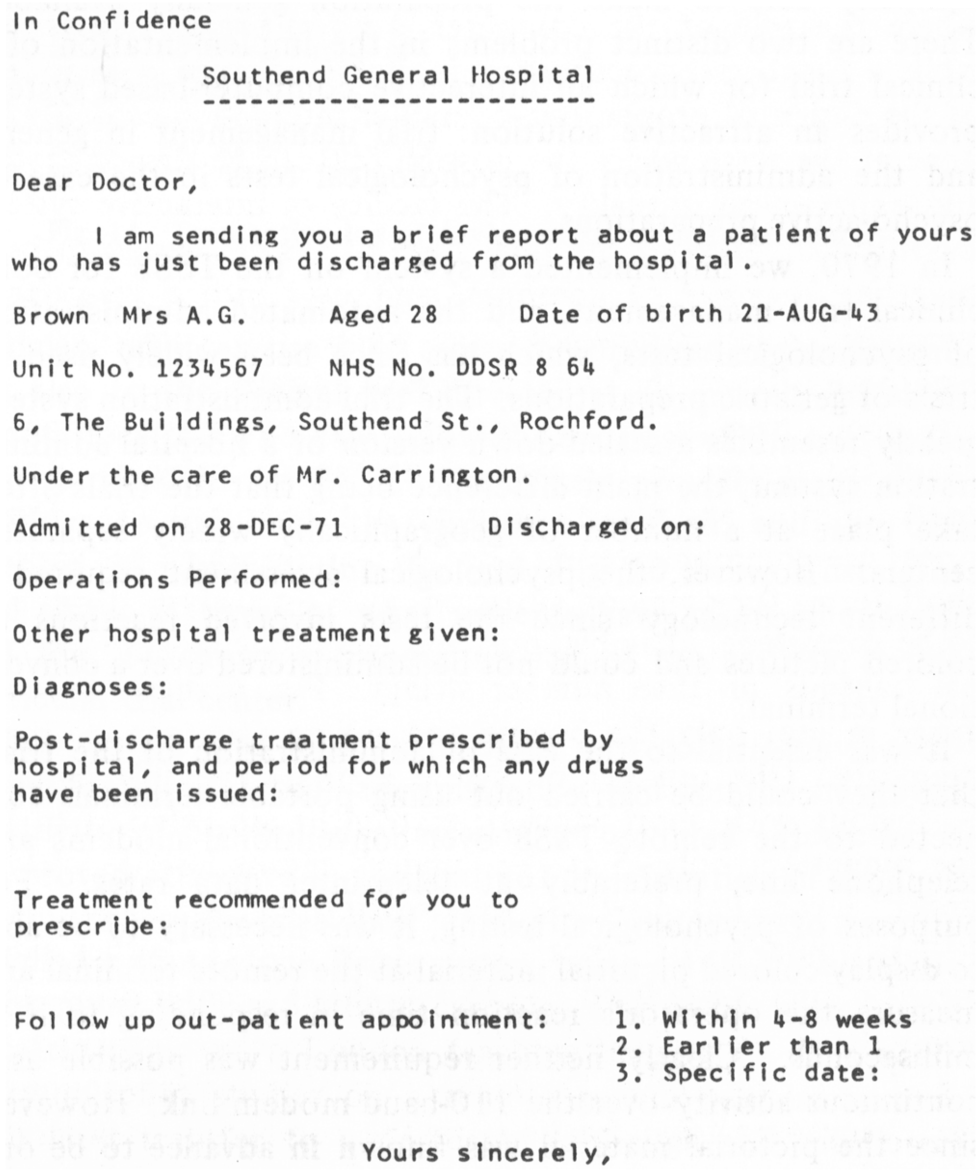
It may seem strange in a paper on interactive systems to make a virtue of avoiding the need for interaction, but it can be overused and the role, and value, of producing documents for circulation is sometimes overlooked. A high-quality document based on the accurate, timely information acquired by the interactive system has a psychological impact of its own. It is businesslike, engendering trust and a desire to maintain its standards, and yet it is not an overwhelming piece of technology for a simple task, as is a visual display. Documents also have the virtue of portability (the catering return goes round the ward with the meals trolley—a visual display would be both cumbersome and splashed with gravy), and can be slipped into the ward records of a patient's case notes.

The length of stay estimates are themselves used to provide documents for use by the consultants to aid their planning of admissions, and by the ward sister to aid her planning of the work pattern of the ward. Fig. 6 shows such a forecast document for one ward—it can be seen that the ward will become overbooked on Monday the 31st according to the current estimates. A further example of document production is the discharge letter shown in Fig. 7, two copies of which are produced when a patient is due for discharge and placed in his case notes. When a consultant decides the patient may leave, he completes the remaining parts of the letter and gives a copy to the patient to present to his doctor. Note again that the consultant does not have to interact with the computer system, and yet it is taking a major part of the workload of letter writing from him.



```
PBS:FORECAST
WARD:BALMORAL

---

MINIMUM EXPECTED BED OCCUPANCY FOR BALMORAL  WARD FROM 29/1/72

DAY     : 1 2 3 4 5 6 7 8 9  10  12  14  16  18  20  22  24  26  28 PATS
----------------------------------------------------------------------------
        :                                                    :
SAT  29:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@19+D           :
        :                                                    :
SUN  30:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@23     :
        :                                                    :
MON  31:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@:@@28+D+D
        :                                                    :
TUE   1:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@:27+D+D+D
        :                                                    :
WED   2:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@24+D+D+D
        :                                                    :
THU   3:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@23+D+D+D+D+D+D
        :                                                    :
FRI   4:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@22+D+D   :
        :                                                    :
SAT   5:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@21+D       :
        :                                                    :
SUN   6:@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@23+D+D :
        :                                                    :
        :                                                    :
DAY     : 1 2 3 4 5 6 7 8 9  10  12  14  16  18  20  22  24  26  28 PATS
----------------------------------------------------------------------------
                                                             :
                                                     (WARD CAPACITY)

EACH  @@  REPRESENTS A PATIENT STAYING OVERNIGHT

EACH  +D  REPRESENTS A DAY-STAY PATIENT OR ONE DUE TO BE
DISCHARGED THAT DAY
```

**Fig. 6. Forecast bed occupancy in one ward.**



In Confidence

Southend General Hospital

Dear Doctor,

I am sending you a brief report about a patient of yours who has just been discharged from the hospital.

Brown Mrs A.G. Aged 28 Date of birth 21-AUG-43

Unit No. 1234567 NHS No. DDSR 8 64

6, The Buildings, Southend St., Rochford.

Under the care of Mr. Carrington.

Admitted on 28-DEC-71 Discharged on:

Operations Performed:

Other hospital treatment given:

Diagnoses:

Post-discharge treatment prescribed by hospital, and period for which any drugs have been issued:

Treatment recommended for you to prescribe:

Follow up out-patient appointment:
1. Within 4-8 weeks
2. Earlier than 1
3. Specific date:

Yours sincerely,

**Fig. 7. Letter to doctor to be completed by the consultant on discharging the patient.**

It is this balanced use of interaction, document production, and human activity, each to best effect in its most useful area, that characterizes a well-designed system. Fortunately the glamour of interactive systems in themselves is wearing sufficiently thin for critical appraisals to be made of the need for proposed interaction. It is also evident in practice that the smaller a system, the more easy it is to make that appraisal correctly.

### *D. A Human Application—Automated Psychological Assessment*

In the case histories given so far, even though the staff involved in interacting with computer have not been computer orientated, they have all been professionally engaged in their own work aided by the computer and might be expected to have an interest in learning to use the system. The present example differs in that it involves the use of terminals by ordinary members of the public, in particular geriatric patients with little previous experience of the use of any technology. The problems involved in automated psychological assessment, and the merits of the technique, are many and varied, but they have been well described by Gedye [3] - [5], [15], [16] and we shall concentrate on the technical problems and results obtained.

When a pharmaceutical company develops a new preparation, it cannot market it without a prolonged series of trials, initially with animals and finally with human patients under clinical supervision. Only after clear evidence of positive therapeutic effects and adequate checks for deleterious side-effects have been presented to a government agency is the company able to make the preparation generally available. There are two distinct problems in the implementation of a clinical trial for which an interactive computer-based system provides an attractive solution: trial management in general and the administration of psychological tests in the case of psycho-active preparations.

In 1970, we implemented a system on the TSS8 for both clinical trial management, and the automated administration of psychological tests, which has since been widely used in trials of geriatric preparations. The trial administration system greatly resembles a scaled-down version of a hospital administration system, the main difference being that the trials often take place at a number of geographically widely separated centers. However, the psychological assessment required a different technology since the tests involved reactions to colored pictures and could not be administered over a conventional terminal.

It was essential to the ease of administration of the trials that they could be carried out using portable terminals connected to the remote TSS8 over conventional modems and telephone line, preferably at teleprinter data rates. For purposes of psychological testing, it was necessary to be able to display colored pictorial material at the remote terminal and measure the operator's reaction time in responding to it in milliseconds. Clearly neither requirement was possible as a continuous activity over the 110-baud modem link. However, since the pictorial material was known in advance to be one out of a set of some hundreds of items, and reaction times of about 400 ms had to be measured only once every four seconds, the actual data rate necessary to control and monitor the interaction was comparatively low.

To take advantage of this low average data rate, it was necessary to select the prepared pictorial material at the remote terminal and to measure the reaction time there also. A terminal was developed for this purpose based on a random-access microfilm projector manufactured by Saab in Sweden [34] and intended for medical and commercial data retrieval [35] . The basic Saab terminal consists of a cabinet similar in size and styling to a 9-in television monitor, containing a microfilm transport and projector for 16-mm film. Recorded on the edge of the film are code bars which are read optically to enable the ends of the film to be located, and individual frames to be located by counting their position from an end. Associated with the projector is an electronic controller which decodes digital control signals as commands to move the film, counts to the required frame, and positions it accurately for display on the screen. The film is transported at a rate of some 200 frames/s which, allowing for acceleration, enables any one of 100 frames to be accessed randomly within a second and any one of 1000 frames to be selected within a few seconds.

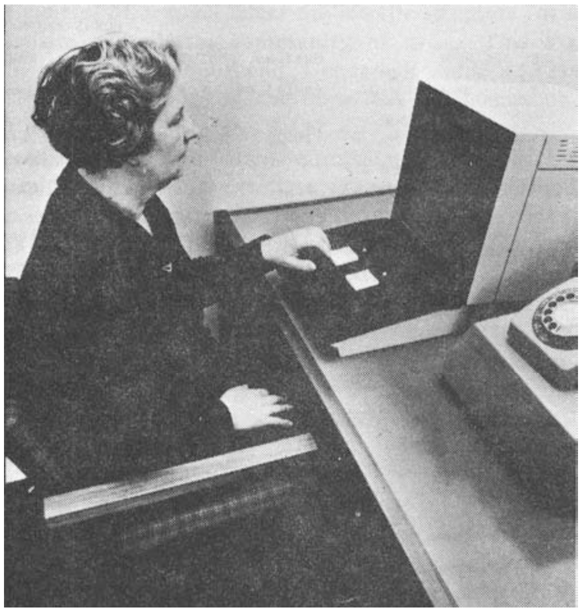
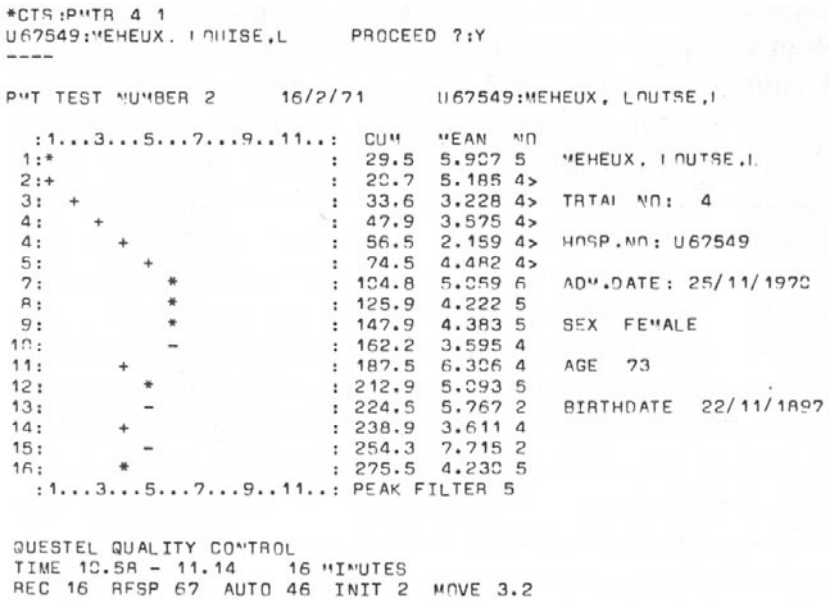


**Fig. 8. Random-access microfilm terminal and reaction-time logger used in psychological assessment.**

The basic terminal was extensively modified both physically and electronically to provide the additional facilities required and to ensure a suitable man-machine interface for use with non-computer-orientated subjects; the final form of terminal is shown in the photograph of Fig. 8. A short pedestal and escutcheon were added to angle the screen comfortably and hide all indicator lamps (whose operation proved to be extremely distracting). The front part of the pedestal also serves as the base for one of a set of keyboard modules, ranging from extremely simple two-key units for psychological testing, to complex multifunction keyboards with indicator lamps for market research studies. In market research applications, the keyboard of the terminal shown in Fig. 6 is replaced by one with a linear array of 15 additional keys near the screen which can be used for multichoice responses and for entering a point on a preference or sujective-rating scale. The pedestal also contains the additional electronics to measure the time between the terminal shutter opening and a key being depressed, and to encode this and the identification of the key into ASCII characters for transmission to the computer.

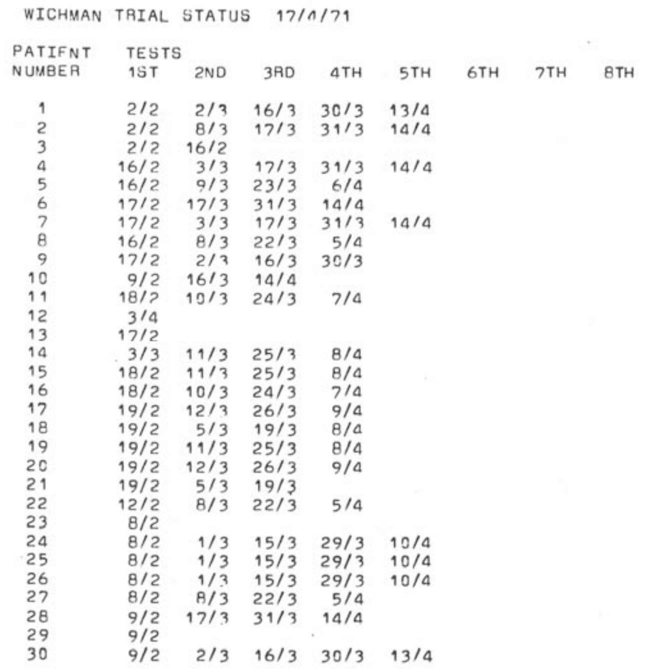
A conventional teleprinter is associated with the microfilm terminal and the complete configuration appears to be a 110baud ASCII teleprinter to the remote computer so that no special arrangements are necessary to drive it. The communications protocol was such that the character stream from the computer is normally directed to the teleprinter until an escape character is received, when it is interpreted as a command to the microfilm terminal. The-command primes the keyboard (where indicator lamps indicate the active keys) and causes the film to be moved. The terminal acknowledges receipt of the command, reports successful movement of the film (the frame counting system has spurious signal and end-of-film detection logic), and transmits back the keys depressed and the reaction time. The ASCII messages involved are readily generated and decoded in Basys.

The microfilm terminal proved to be totally acceptable to patients and readily used even by institutionalized geriatric patients who were otherwise regarded as completely incapable. A complete analysis of the results of the test is immediately available on the teleprinter when the patient has completed it; Fig. 9 shows such an analysis indicating graphically the trajectory through the test, the maximum level reached, time taken, and other statistics. This result is available not only to the clinician, who can use it to assess the progress of the patient, but also to the clinical trial director at an entirely different remote location who can follow the progress of the trial at his own terminal to the TSS8. The system provides many other facilities to aid both the clinicians involved, and the trial director, in administering the trial; Fig. 10 shows a listing of trial status in terms of what patients have taken what tests, and Fig. 11 shows a transaction log of the activity at one particular trial center.



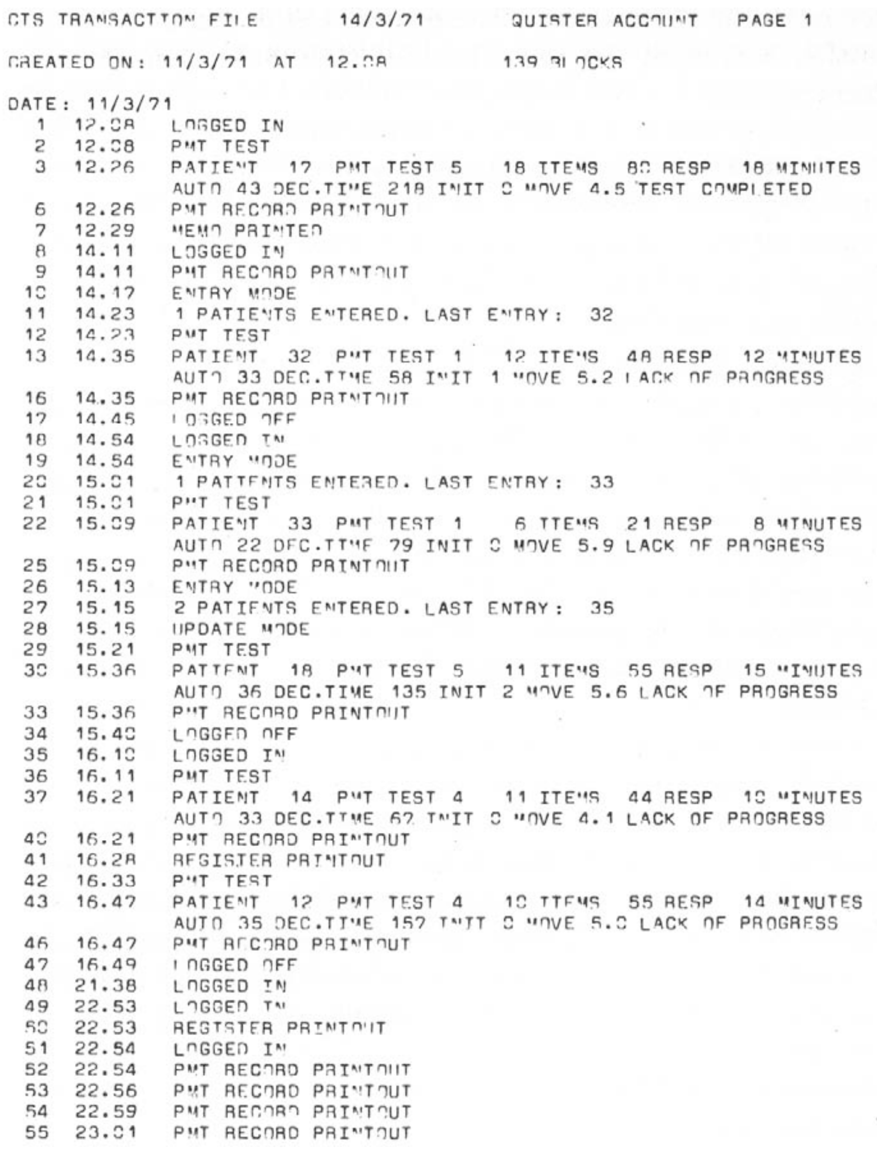
```
*CTS:PMTR 4 1
U67549:MEHEUX. LOUISE,L      PROCEED ?:Y
----

PMT TEST NUMBER 2      16/2/71       U67549:MEHEUX, LOUISE,I

  :1...3...5...7...9..11..:  CUM   MEAN  NO
 1:*                      :  29.5  5.907 5   MEHEUX, LOUISE,I
 2:+                      :  20.7  5.185 4>
 3:  +                    :  33.6  3.228 4>  TRIAL NO:  4
 4:    +                  :  47.9  3.575 4>
 4:      +                :  56.5  2.159 4>  HOSP.NO: U67549
 5:        +              :  74.5  4.482 4>
 7:          *            : 104.8  5.059 6   ADM.DATE: 25/11/1970
 8:          *            : 125.9  4.222 5
 9:          *            : 147.9  4.383 5   SEX  FEMALE
10:          -            : 162.2  3.595 4
11:      +                : 187.5  6.306 4   AGE  73
12:        *              : 212.9  5.093 5
13:        -              : 224.5  5.767 2   BIRTHDATE  22/11/1897
14:      +                : 238.9  3.611 4
15:        -              : 254.3  7.715 2
16:      *                : 275.5  4.230 5
  :1...3...5...7...9..11..: PEAK FILTER 5


QUESTEL QUALITY CONTROL
TIME 10.58 - 11.14    16 MINUTES
REC 16  RFSP 67  AUTO 46  INIT 2  MOVE 3.2
```

**Fig. 9. Analysis of performance on computer-administered test.**



```
WICHMAN TRIAL STATUS  17/4/71

PATIENT    TESTS
NUMBER     1ST    2ND    3RD    4TH    5TH    6TH    7TH    8TH

   1       2/2    2/3   16/3   30/3   13/4
   2       2/2    8/3   17/3   31/3   14/4
   3       2/2   16/2
   4      16/2    3/3   17/3   31/3   14/4
   5      16/2    9/3   23/3    6/4
   6      17/2   17/3   31/3   14/4
   7      17/2    3/3   17/3   31/3   14/4
   8      16/2    8/3   22/3    5/4
   9      17/2    2/3   16/3   30/3
  10       9/2   16/3   14/4
  11      18/2   10/3   24/3    7/4
  12       3/4
  13      17/2
  14       3/3   11/3   25/3    8/4
  15      18/2   11/3   25/3    8/4
  16      18/2   10/3   24/3    7/4
  17      19/2   12/3   26/3    9/4
  18      19/2    5/3   19/3    8/4
  19      19/2   11/3   25/3    8/4
  20      19/2   12/3   26/3    9/4
  21      19/2    5/3   19/3
  22      12/2    8/3   22/3    5/4
  23       8/2
  24       8/2    1/3   15/3   29/3   10/4
  25       8/2    1/3   15/3   29/3   10/4
  26       8/2    1/3   15/3   29/3   10/4
  27       8/2    8/3   22/3    5/4
  28       9/2   17/3   31/3   14/4
  29       9/2
  30       9/2    2/3   16/3   30/3   13/4
```

**Fig. 10. Status of trial. For each patient, the dates of the tests they have taken.**



```
CTS TRANSACTION FILE        14/3/71          QUISTER ACCOUNT    PAGE 1

CREATED ON: 11/3/71  AT   12.08             139 BLOCKS

DATE: 11/3/71
  1   12.08    LOGGED IN
  2   12.08    PMT TEST
  3   12.26    PATIENT   17  PMT TEST 5    18 ITEMS   80 RESP   18 MINUTES
               AUTO 43 DEC.TIME 218 INIT 0 MOVE 4.5 TEST COMPLETED
  6   12.26    PMT RECORD PRINTOUT
  7   12.29    MEMO PRINTED
  8   14.11    LOGGED IN
  9   14.11    PMT RECORD PRINTOUT
 10   14.17    ENTRY MODE
 11   14.23    1 PATIENTS ENTERED. LAST ENTRY:  32
 12   14.23    PMT TEST
 13   14.35    PATIENT   32  PMT TEST 1    12 ITEMS   48 RESP   12 MINUTES
               AUTO 33 DEC.TIME 58 INIT 1 MOVE 5.2 LACK OF PROGRESS
 16   14.35    PMT RECORD PRINTOUT
 17   14.45    LOGGED OFF
 18   14.54    LOGGED IN
 19   14.54    ENTRY MODE
 20   15.01    1 PATIENTS ENTERED. LAST ENTRY:  33
 21   15.01    PMT TEST
 22   15.09    PATIENT   33  PMT TEST 1     6 ITEMS   21 RESP    8 MINUTES
               AUTO 22 DEC.TIME 79 INIT 0 MOVE 5.9 LACK OF PROGRESS
 25   15.09    PMT RECORD PRINTOUT
 26   15.13    ENTRY MODE
 27   15.15    2 PATIENTS ENTERED. LAST ENTRY:  35
 28   15.15    UPDATE MODE
 29   15.21    PMT TEST
 30   15.36    PATIENT   18  PMT TEST 5    11 ITEMS   55 RESP   15 MINUTES
               AUTO 36 DEC.TIME 135 INIT 2 MOVE 5.6 LACK OF PROGRESS
 33   15.36    PMT RECORD PRINTOUT
 34   15.40    LOGGED OFF
 35   16.10    LOGGED IN
 36   16.11    PMT TEST
 37   16.21    PATIENT   14  PMT TEST 4    11 ITEMS   44 RESP   10 MINUTES
               AUTO 33 DEC.TIME 67 INIT 0 MOVE 4.1 LACK OF PROGRESS
 40   16.21    PMT RECORD PRINTOUT
 41   16.28    REGISTER PRINTOUT
 42   16.33    PMT TEST
 43   16.47    PATIENT   12  PMT TEST 4    10 ITEMS   55 RESP   14 MINUTES
               AUTO 35 DEC.TIME 157 INIT 0 MOVE 5.0 LACK OF PROGRESS
 46   16.47    PMT RECORD PRINTOUT
 47   16.49    LOGGED OFF
 48   21.38    LOGGED IN
 49   22.53    LOGGED IN
 50   22.53    REGISTER PRINTOUT
 51   22.54    LOGGED IN
 52   22.54    PMT RECORD PRINTOUT
 53   22.56    PMT RECORD PRINTOUT
 54   22.59    PMT RECORD PRINTOUT
 55   23.01    PMT RECORD PRINTOUT
```

**Fig. 11. Transaction log of day's activity at one trial center.**

This system was initially justified as a cost-effective technique for clinical trial administration and for the improved sensitivity of psychological assessment provided. However, it has proved attractive in the routine administration of the type of clinic where the trials have taken place, and we have recently installed a stand-alone Basys system using a 16K × 8 bit Minic I computer with a 5-Mbyte cartridge disk as file store in the geriatric clinic of a London teaching hospital (supported by a grant from the Nuffield Founadtion). This possibility of pilot studies on a remote general-purpose system, with later transfer to a dedicated (although still multiuser) system, has proven important in many applications in allowing design and development to proceed with full user interaction before there is any commitment to hardware.

## *E. Some Other Applications*

There has been a wide range of Basys system applications similar to those already described, e.g., the gilt-edged securities system has its counterparts in foreign exchange broking systems;

and the terminal for automated psychological testing has proved equally efficacious in administering market research questionnaires. Once Basys was available, however, it became attractive to use it in a range of applications where user interaction was not of prime importance, but where the compactness, power, or modularity of the implementations was of greater interest. Some of these applications throw additional light on the techniques we have described and are reviewed briefly in this section.

An important class of applications is that where a minicomputer is to be used to control a complex peripheral device and, rather than write all the applications in machine code, an implementation of Basys has been extended to interface the control programs to a Basys command. A good example is a parametric speech synthesizer whose control program was incorporated into Basys on the PDP9. All the programs to translate between phoneme/ inflection strings and synthesizer parameters are written in Basys. They generate an array which is passed to a small machine-code synthesizer control program, and this outputs the contents of the array to the synthesizer. A similar approach has been taken to instrumentation where, for example, a mass spectrometer interfaced to a Minic I has its own machine-code data acquisition and smoothing programs, but all the calibration and analysis programs are written in Basys. In another signal-processing application, an FFT command has been incorporated in Basys which applies a machine-code Fourier transform program to a Basys array.

An application which made full use of both interaction and extendability was an emulator for the Minic I machine in a real-time application [8]. We had to develop a patient monitoring system for an intensive care ward using a 1024 × 8 bit Minic coupled to a 128-kbyte drum, graphic display, clinician's keyboard, teleprinter, tape reader, magnetic tape, and an analog/digital convertor multiplexed around up to 64 transducers (8 beds × 8 transducers). The target machine was too small for software development and many of the peripherals were themselves being developed. We decided, as an experiment, to develop the complete system under an emulator, not only of the computer, but also of the peripherals and interrupt system.

A command, Minic, was added to a PDP10 implementation of Basys which picked up an array containing a block of parameters, the values of the Minic hardware registers, and the 1K × 8 bit core store. A machine code emulator then executed a number of Minic instructions and returned to Basys. It also returned when it encountered input/output or microprogram extension entry instructions, allowing all peripheral transfers to be mediated by Basys. Emulators for all the peripherals on the system were written in Basys itself, as was a symbolic debug program. The Minic simulator kept track of instruction execution times so that clock interrupts could be emulated. Other interrupts could be emulated by passing control to the Minic simulator for randomly distributed time periods. The complete emulation was available as an interactive program in Basys on the.PDP10 and the programs developed for the target system were debugged from a terminal. The final patient-monitoring system consisted of some 30 256 × 8 bit machine code overlays.

An assembler for the normal Minic assembly language was written in Basys in about two man-days. The symbolic debug and emulation environment took about three man-days. The same system has also been used to develop a floating-point arithmetic package and a Basys interpreter for other Minic computers. The ease of these developments indicates that the use of an emulator imbedded in a high-level language on an interactive system is a very effective technique for both normal software and real-time system development.

The string-handling facilities of Basys have made it attractive to many users requiring nonnumeric data processing, for example, in the analysis of literary texts, grammatical inference, and so on. Its simplicity of use has also caused it to be widely used where other, more efficient, numerical data processing languages would appear more appropriate, e.g., statistical studies, clustering techniques, automata analysis, etc. The ease of making these packages available in interactive form has itself proved important and is leading to the development of man-machine collaborative systems even when not originally intended.

## III. THE PROGRAMMING OF INTERACTION

When one examines the technical factors underlying the wide range of successful applications of interactive minicomputer systems programmed in Basys, the language itself and its implementation come naturally to mind. However, there is another technology which plays an equal, if not greater, part and which is so novel in concept that it is easily overlooked. This is the programming of the interactive dialogue between user and system to ensure simple, natural, and accurate man-computer communication. The literature on this topic is sparse, poor features of early designs are propagated, and useful techniques too rarely publicized. Martin's recent book [36] is particularly important in recognizing the subject area and providing a wealth of examples from operational systems. Kennedy [37] has investigated experimentally the behavior of non-computer-orientated staff using a Basys system in a hospital, and has argued for certain fundamental techniques in establishing interactive dialogues [38]. There are also the beginnings of analytical studies of interaction [39].

It would be wrong to pretend to any science in the development of effective interactive dialogues at present. However, as a result of trial and error in our own experience of developing interactive systems, we have come to regard certain techniques as extremely valuable. The following sections outline these techniques, discuss their rationale, and summarize them as a set of useful "rules."

### *A. The User Image of the System*

The most important single consideration in programming interaction is the user's image of the system. Sitting at a teleprinter or visual display interacting with a remote machine, remote from other users and the results of his interaction, the system user forms some model of what lies behind his terminal. This model can be very complex indeed—we have shown elsewhere that a system as simple as a two-state stochastic automaton leads to indefinitely complex models under certain, not unrealistic, experimental conditions [40]. A major objective of the system designer must be to avoid completely all unnecessary complexity in the user's model. It is not easy to be precise, however, about what is desirable—a user at a terminal is not just another automaton but a human being subject to emotion, motivation, learning, and so on. He will react to the "personality" of the system as much as its function and, with users whose motivation to become involved with a computer is not high, this may be of prime importance.

What factors influence the user's image of the system and how does it come to be built up? We may distinguish the following characteristics.

1) *Stereotyped Expectations about Computers*: The public image of computers has quite strong stereotypes which affect a user's advance expectations. It may be worthwhile to use questionnaires to evaluate these explicitly before deciding how to represent the system to

prospective users. However, it has been our experience that these expectations are modified - rapidly as a result of experience. It is important to show prospective users a system in action and let them get the feel of good interactive dialogue before evaluating their own assessments of how they feel about using one.

*Rule 1—Introduce through experience: Interactive systems are meant to 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.*

2) *Expectations Based on Past Experience of Computers*: Expectations that have a firmer foundation than general knowledge are more serious. Again most individuals seem to adapt rapidly, particularly to an improved situation. However, the overlearning of specific operational procedures on one system may interfere strongly with superficially similar procedures on another. This problem is not unique to users of computer systems and difficulties with relearning are to be expected. Where one can help is by giving immediate feedback on the results of actions so that the resultant negative reinforcement is at its most effective. A user who performs an action expecting a certain result will continue to make the error for a long period if the required result has apparently been attained, even if its attainment is a negative inference from nothing going wrong— yet! It is certainly easier to work with totally naive users than with non-computer-orientated staff who have had some experience of another interactive system. However, much may be done to aid relearning, and certainly much may be done that hinders it.

*Rule 2—Immediate feedback: Give the user feedback by making an immediate unambiguous response to each of his inputs. This should be sufficient to identify the type of activity taking place.*

For example, the following sequences illustrate the generation and resolution of confusion for a user of a television advertisement booking system who has previously used the command E to enter a booking, but whose present system uses it to enter a new time slot.

*Poor Sequence*

```
COMMAND: E
DATE: 7 MAY
TIME: 2054
DURATION: 20
SLOT EXCEEDS MAXIMUM FREE TIME* OK?: NO
COMMAND: I
```

and so on. It was only when he received the unexpected warning, impossible in a booking sequence, that the user became aware, indirectly, that he was in a slot entry, rather than booking entry, sequence. The system gave him the opportunity to abort the activity but only after he had made three, apparently correct, responses. Such a sequence both causes frustration and partially reinforces incorrect behavior.

*Good Sequence*

```
COMMAND: E
NEW SLOT DATE::
COMMAND:I
```

and so on. The user just typed colon when he saw NEW SLOT DATE and realized immediately he had started the wrong activity. The system recognizes this as an abort command (see text referring to Fig. 14) and terminates the activity.

3) *Expectations Based on Task*: The expectations of how interaction will proceed based on a person's past experience of the same task not involving computers are by far the most important. In many cases they are based, not merely on past behavior patterns, but rather on a logical model of a group of related activities, e.g., that any human interrogator would ask about A and B before C because he has to perform an auxiliary calculation with A and B first. Worse still is the case where the human would not request information about C once he knows that A and B have certain values. In this case, it goes against the whole logic of the task for C to be requested first, whereas the equally redundant request of C last can be more readily accepted as an irrelevant, but not illogical, request.

Task-based expectations are the most powerful and long lasting, and it is vital that the interactive dialogue does not appear to be based on a conflicting model. We have found it useful to pose the question, "How would a colleague ask you for this information if you were discussing the same activity?" There is a danger that the system analyst will structure the activities in a framework that is completely logical and selfconsistent, but one that is a hitherto unnoticed alternative to the conventional user stereotypes of the same activities. There is a worse danger that the system designer will evaluate this framework as better and seek to impose it on the user.

*Rule 3—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.*

A simple example here is the request for the sex of a patient in a clinical system. If the patient's full name is known, in the majority of cases a person will not need to ask the sex, and hence a Name?, Sex?, sequence appears illogical. However, particularly when admitting a patient, it is natural to ask for the sex first. Clearly the name cannot be inferred from this, and a Sex?, Name?, sequence does not appear illogical.

4) *Expectations Generated by Interaction with the System*: Once a user has begun to interact with a system he begins to build up expectations based on generalization from his experience. This phenomenon can be very valuable if correctly used since it enables operational training based on one part of the system to be transferred to the whole. On the other hand, it can greatly undermine a user's confidence in the system, and his own ability to handle it, if strong expectations based on previous experience are false predictors of what actually happens. Those who have had such experiences as the interactive editor and calculator, from the same manufacturer for the same machine, which use different control characters for delimiters, respond in a different way to erase character, eraseline, messages, etc., will have every sympathy with the naive user of a new system faced with similar problems.

*Rule 4—Consistency and uniformity: Ensure that all terminology and operational techniques are consistently applied, and uniformly available, throughout all system activities.*

An example here is the provision of a system facility such as a memory prompt when the user is not sure what to do. The same message at any level should give the relevant information at that level, e.g., a poor implementation is the use of the character H as a request for help at one level and question mark ? at another.

## *B. Learning, Teaching, and User Control*

The preceding generalities about the user image of the system, and the basis for its expectations, lead to some specific techniques for programming interaction. There are other considerations of user psychology which are equally important, particularly the roles of learning, the system's part in teaching, and the degree of control afforded the user.

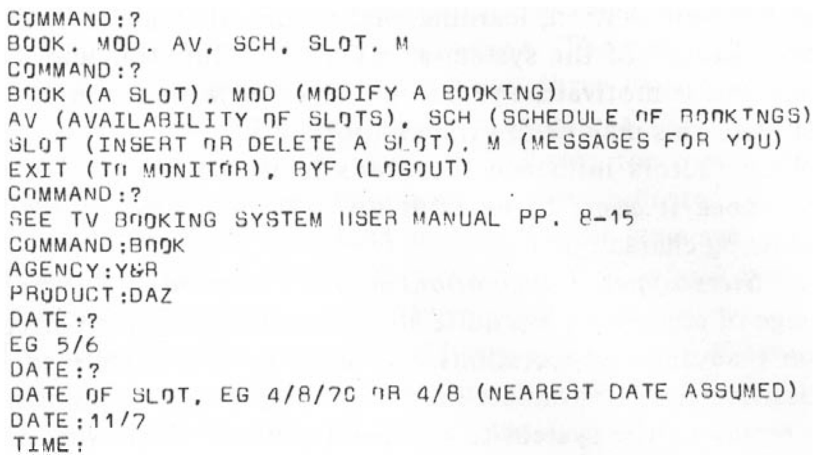
The first-time user of a system needs all the help he can be given and the system messages need to be long enough to be explicit and unambiguous. An infrequent user of the system may rely on this help and never learn to react to less explicit cues. Continuous users with increasing operational experience, however, can come to be irked by the very verbosity which was such a help to them originally. In close human partnership, the channel of communication becomes more and more efficient, messages become terser and more encoded, and both sides of the partnership adjust to one another.

We have made many clumsy attempts to emulate this process of joint adaptation in our design of interactive dialogues, and would warn against too subtle methods. Changes in the behavior of the system, apparently taking place without cause and of its own volition, are among the most confusing phenomena possible. We remarked earlier, as does Kennedy [37], that the variable delays normally occurring on a time-sharing system are a source of stress, and basic system-theoretic considerations [40] indicate why any such phenomena create difficult problems for the user in modelling the system. This leads to a further rule before considering changes to compensate for the user's learning.

*Rule 5—Avoid acausality: Do not introduce apparently acausal phenomena into the system. Make changes in the system clear consequences of the user's actions.*

A good warning example here is the use of timing-out responses to "help" tardy users by prompting them. We implemented this as an obvious and technically simple device on one early system, but had to remove it in haste after an otherwise stable user became highly agitated and refused to go near a terminal again after her first timed-out prompt. She had been quietly thinking what to do and the terminal suddenly interjecting and making its own suggestions was just too much for her. In general, users do think of the computer as a tool, not as an equal, although they often seem to expect remarkable perspicacity in the tool—the image of a good servant probably most clearly expresses this ambiguous role.

Having emphasized that changes in the system's mode of behavior must be under user control, how does one suggest that compensation be made available for the user's increasing experience? There are two specific techniques that we have found of great value which between them seem able to cope with most requirements. These are query-in-depth, and parallel/sequential tradeoff.



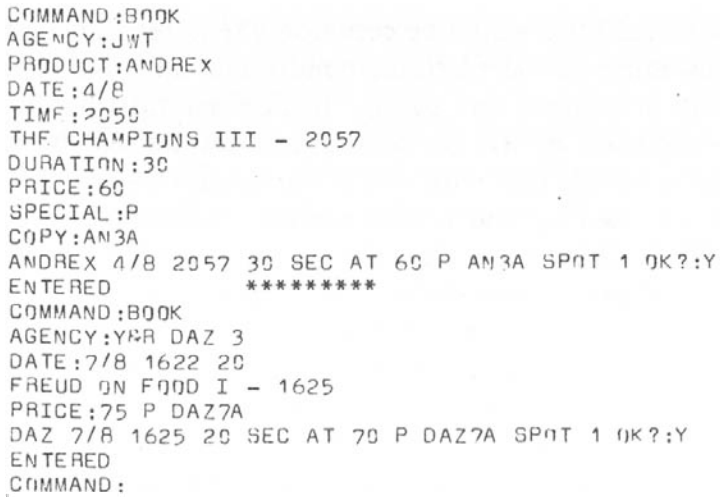
```
COMMAND:?
BOOK. MOD. AV, SCH, SLOT. M
COMMAND:?
BOOK (A SLOT), MOD (MODIFY A BOOKING)
AV (AVAILABILITY OF SLOTS), SCH (SCHEDULE OF BOOKINGS)
SLOT (INSERT OR DELETE A SLOT), M (MESSAGES FOR YOU)
EXIT (TO MONITOR), BYE (LOGOUT)
COMMAND:?
SEE TV BOOKING SYSTEM USER MANUAL PP. 8-15
COMMAND:BOOK
AGENCY:Y&R
PRODUCT:DAZ
DATE:?
EG 5/6
DATE:?
DATE OF SLOT, EG 4/8/7C OR 4/8 (NEAREST DATE ASSUMED)
DATE:11/7
TIME:
```

**Fig. 12. Query-in-depth in use on a TV booking system. Each successive question mark entered at a given stage elicits more detailed information.**

The query-in-depth technique is very simple, but in this very simplicity lies its success since it is rapidly assimilated by naive users. Essentially, at any point where the user has to make a response requested by the system he can type a question mark ?. The system then responds with a list of possible responses the user might make or, if data entry is required, gives an illustration of data in the appropriate format for entry. This query facility is itself valuable, but it can be defined by allowing the user to continue to type in question marks and giving more information in response to each one, i.e., allowing questioning in depth. In particular, the first response can be a memory jogging brief form of the most likely responses, such that even an impatient experienced user will not hesitate to use it for fear of provoking a massive explanatory printout. The second response can be a full-form list of all possible responses including aborts, etc., and the third level either a section of the operator's manual or, particularly with a slow terminal, a reference to the appropriate pages in the manual. It can be seen that the query technique effectively distributes the operator's manual through the system in an optimal fashion, enabling the system itself to assume a training role, but retains absolute user control over the system's verbosity. It has the clear advantage that a user of only part of the system gains easy access to the relevant information for that part and does not have to find his way through a thick and daunting manual. We have found that it virtually removes the need for explicit operator training, and that the manuals fall into disuse when this is implemented. One further refinement is that the data entry examples should be generated as requested so that successive requests lead to different examples and avoid users being misled by particular figures (the acausality here does not matter because the user sees an apparent cause in his own repeated requests). Fig. 12 illustrates query-in-depth in action.

*Rule 6—Query-in-depth: Distribute information and tutorial material appropriately throughout the system to be accessed by the user through a simple uniform mechanism.*

The sequential/parallel tradeoff technique is complementary to the query technique since it allows the user to reduce still further the system's verbosity and his own associated activity as he comes to remember more of a data entry sequence. Essentially, when a user has to enter a series of commands or data items sequentially, he is permitted to enter any number of them on one line, in the same sequence separated by spaces. This gives the user a high degree of control over the structure of his data entry sequence and, with increasing skill, he can group items together as a single entry. The growth of such conceptual groupings is a natural part of the learning process, and the effect to the user is very much one of mutual adaptation. There are more technical problems with sequential/parallel tradeoff than with query-in-depth since one Whist not allow ambiguity to arise, and does not wish to reduce the system's error detecting capability. It is necessary to design the response formats to be separable when several responses are strung together, and to attempt to make the syntax of different responses sufficiently different to detect errors in the parallel entry format. Fig. 13 shows a user first entering his data completely sequentially and then splitting it into conceptually sensible groups for parallel entry. Note that this is totally at the user's instigation.



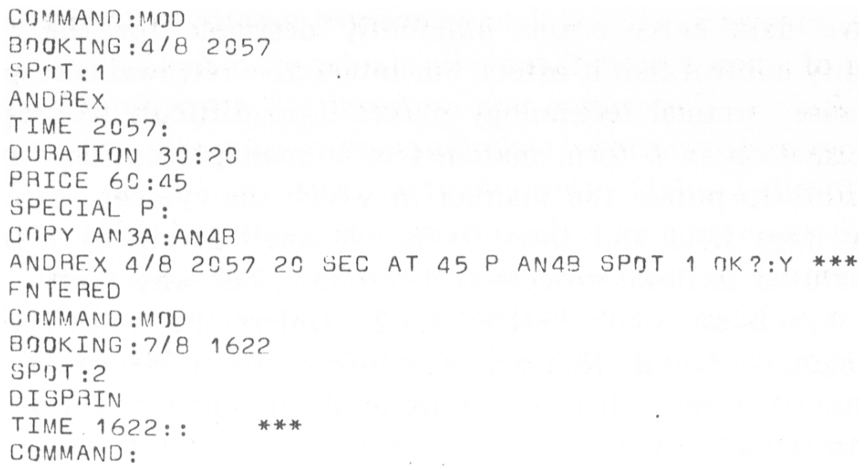
```
COMMAND:BOOK
AGENCY:JWT
PRODUCT:ANDREX
DATE:4/8
TIME:2050
THE CHAMPIONS III - 2057
DURATION:30
PRICE:60
SPECIAL:P
COPY:AN3A
ANDREX 4/8 2057 30 SEC AT 60 P AN3A SPOT 1 OK?:Y
ENTERED          *********
COMMAND:BOOK
AGENCY:YKR DAZ 3
DATE:7/8 1622 20
FREUD ON FOOD I - 1625
PRICE:75 P DAZ7A
DAZ 7/8 1625 20 SEC AT 70 P DAZ7A SPOT 1 OK?:Y
ENTERED
COMMAND:
```

**Fig. 13. Parallel/sequential tradeoff under user control in data entry. In the sequence up to the asterisks, the user makes the booking one item at a time with individual prompts from the program. In the following sequence, he makes a similar booking entering several items at a time with a prompt only for the next unentered item.**

*Rule 7—Sequential-parallel tradeoff: Allow the user maximum flexibility to make his responses holistically (in parallel) or serially (in sequence) according to his wishes.*

Our emphasis on user control of the modes of operation suggests a control-theoretic analogy, and we have found it useful to think of the user as the observer and controller of an automaton. This automaton should be designed to be observable so that the user is always completely aware of its location on its state diagram, and it should be controllable so that the user can readily take it into whatever state he requires. In particular we have found it worthwhile to standardize on another activity-wide response, the abort response, which terminates the current activity and returns the user to a standard initial state, i.e., makes the automaton he is controlling resettable. The implementation of this facility requires care in performing sequential updates, but it is a good software discipline not to take account of an update until it is complete, and it is vital to the user that he can abort an erroneous activity cleanly and completely. We use the colon, :, as an abort request character because it always terminates the system's request for input, and users find it easy to remember, "the colon means the system wants a response from you—type it back to it if you want to get back to the start." The colon was also chosen as being a non-shift character easily entered on most terminals. It is surprising how many otherwise excellent systems use delimiters which are physically more difficult to access such as backarrow ←. It is elementary ergonomics to minimize the effort of typing.

One further contribution to observability is to print out the present value of a variable when an updated value is requested from the user. This also acts as a teaching facility since it enables the expected input format to be displayed. The use of these facilities may be seen in Fig. 14 where the user first modifies one booking, but then on the second occasion realizes that he has picked out the wrong product and aborts the sequence.



```
COMMAND:MOD
BOOKING:4/8 2057
SPOT:1
ANDREX
TIME 2057:
DURATION 30:20
PRICE 60:45
SPECIAL P:
COPY AN3A:AN4B
ANDREX 4/8 2057 20 SEC AT 45 P AN4B SPOT 1 OK?:Y ***
ENTERED
COMMAND:MOD
BOOKING:7/8 1622
SPOT:2
DISPRIN
TIME 1622::     ***
COMMAND:
```

**Fig. 14. Observability and resettability of system. At the first asterisks the program prints out a complete entry so that the user can observe the effect of his request. At the second asterisks, the user decides to abort the sequence and gives a standard reset response which takes the system back to its initial state when he commenced the sequence.**

*Rule 8—Observability and controllability: Envision the system as an automaton controlled by the user and make it simple to observe and control. In particular, provide a "reset" command which aborts the current activity cleanly, and print out the current state of a variable when requesting a new value.*

### *C. Other Considerations in Programming Interaction*

It is not necessary to emphasize the role of the computer in data validation since improved quality of data is generally the main justification for the use of interactive data entry. Checking the syntax of the input is generally safe and often very useful since entering the wrong type of data is a common error. It aids this process, and helps the user's memory, to vary the syntax for different items, e.g., 10:7 for a time but 10/7, or 10-JUL, for a date. Checking values on entry can be dangerous if applied over enthusiastically since it is often surprising how much normal data are outside the proposed "norms"! We have become very wary of ever allowing the system to refuse a data item because it is outside a norm, and even having it query items should be done with care since there is great frustration in having to say, "yes I really did mean it," too often. A useful technique, which takes full advantage of human visual-pattern perception, is to output the data input during an entry sequence as a neatly formatted printout at the end of the sequence and ask, "OK.to enter this?" This does not interrupt the operator's train of activity, and is a natural break at which to check what he has done. Examples of this form of validation are given at the end of the booking sequences in Fig. 13.

*Rule 9—Validation: 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.*

One may easily become oversold on the splendors of interaction in interactive systems and assume its use for activities in which it is not necessary and where the limitations of current terminals are most apparent. In particular, one may neglect to take full advantage of the rapid scanning mechanisms of human visual perception for information retrieval, and of the portability and versatility of printed documents themselves. Interactive data entry seems inherently desirable for the data quality control that it offers, but interactive retrieval is suspect because terminal technology makes it so difficult to present information in a form matched to human perception. It is possible to mimic the manner in which the eye can search a very large field and then fixate on small details by a tree-structured retrieval process at a terminal, but why do it when the eye is so much better at it? Unless the data base is changing so rapidly that only absolutely current information is of use, it is better to concentrate on document production for information retrieval and dissemination. Most systems are geared to document circulation before the introduction of a computer and it is unwise to assume that it is necessary or desirable for interactive terminal access to replace them. Basys has extensive facilities for document formatting, and we have found the production of 'secretarial' quality documents a major feature in all applications, e.g., Figs. 5 and 7.

*Rule 10—Use documents: Do not assume that all users need to interact with the system. In particular, use high quality document production rather than terminal interaction for information retrieval whenever possible.*

One advantage of performing any activity on a computer is that it is possible to keep a log of what occurs and process this to give data on the activity at any level of detail. It is easier to accumulate too much information rather than too little, and we have found a log of major activities, query requests, and errors detected is adequate for most purposes. While the system is still at an evaluation and design stage, this, combined with looking-over-the-shoulder, provides feedback on systems operation. Later it becomes feedback on the problems of individual users. Computers are excellent monitoring systems and can keep track of their own system design faults, and the behavior of their users, both sources of research data so vital to the future development of interactive systems, e.g., Fig. 11.

*Rule 11—Log activities: Use the computer to maintain records of system and user activities to evaluate the behavior of both system and users.*

We have proposed a number of specific techniques for programming interaction not as absolute tenets (rules are made to be broken), but as useful guidelines based on experience. There is clearly a major need, and great scope, for research studies of interactive dialogues. It is possible to gather the data for these studies on any interactive system, and it is worth considering such monitoring as an integral part of any system design.

## IV. MAIN FEATURES OF BASYS AND ITS IMPLEMENTATION

All the applications described and the interactive programming techniques suggested have been developed in the interactive language Basys or one of its commercial variants. However, as noted previously, we do not intend to present Basys as a language in this paper. This has been done elsewhere [26] and, in most respects, Basys is yet another problem-orientated language. In this section, we discuss briefly a few selected aspects of Basys and its implementation that seem particularly important to the results achieved and have general relevance for interactive system development.

### *A. Aids to Interactive Programming*

Initially, we did not intend to develop a general-purpose language but rather one aimed at certain concepts of man-computer interaction. However, it became apparent that certain basic facilities would be commonly required in all systems, such as numeric calculations, conditional and unconditional transfers of control, and so on. In our investigations of interactive facilities in use on central utilities we had been very impressed by the ease with which Dartmouth Basic [13], [14] was being used by the non-computer-orientated community. It appeared to have an essential simplicity and naturalness in syntax and editing procedures that commended it to users who were averse to other computer languages. In these circumstances, it seemed unreasonable to invent yet another language structure and, like many other groups developing specialist languages, we decided to model our overall system on Basic.

The two main features of the Basic syntax are its use of meaningful key words and the use of line numbers rather than labels. The key words seem to contribute much to Basic's high readability, and the need to insert them itself prevents the vast opaque syntactic constructions possible in Algol and mandatory in Lisp. The role of the line numbers and their use in transfers of control is more open to controversy, and we have debated it on many occasions over many years, particularly in relation to more meaningful labels. However, the situation of a programmer at a terminal is that he cannot see all his text at a time but must access individual lines. The topology of the number system is so ingrained in us all that numeric labels corresponding to the order of statements give an immediate picture of the stored program structure. Using the same line numbers that make for ease of editing as the "labels" for transfer of control is then a minimal natural construct necessitating the acquisition of no new concepts. We accepted the use of line numbers in Basic and have not regretted it. Indeed, in Basys, it is extended to enable dynamically varying character strings and string constants to be stored and referenced as part of the same structure of numbered "program" lines.

The main objectives of our syntactical modifications to Basic were to minimize the effort of program creation and documentation and maximize the clarity of the result. The subobjectives were the following.

1) No unnecessary syntax on program entry—the programmer should be able to enter the minimum string necessary to specify a statement.

2) Full clarity in program listings—the system should recreate missing syntax on output and format it appropriately.

To achieve this, we had to drop some Basic conventions, notably the non-significance of spaces which were natural separators, readily inserted with the space bar. This allowed a comma, or one or more spaces, or implicit separation, to be specified as optional separators. Command key words could then be defined as a string of letters which matched, or partially matched, one of the stored commands. Hence, any command could be shortened to its minimal unambiguous initial string. We chose command names so that the first two letters were sufficient to resolve ambiguity, and a single letter alone, if ambiguous, was interpreted as the most commonly used command. These abbreviations apply only to program input and, when the whole or part of a program is listed, the full command words are output with a leading and a trailing space to give a neatly formatted, readable output.

The implementation of Basys has always been either interpretive or has retained sufficient information at runtime to appear so to the user. When program execution is halted by use of a control-key, by a stored command, or by an error, any command may be directly executed, and all the data structures are available. Hence, changes may be made to program or data and execution continued.

None of these features is unique to Basys, but taken as a whole, together with a consistent attention to ease of program development in other parts of the system, they provide an attractive and helpful environment to the system developer. How important this is never becomes apparent until a Basys user has to use normal Basic or some other language and finds that “obvious” and “natural” features and facilities do not exist.

### *B. String Processing in Basys*

Our early experiences in programming interactive dialogues had convinced us of the need for facilities to process character strings which went well beyond those of any commonly available language. Our ojectives were very similar to those described by Griswold for Snobol [24] and some striking similarities between the facilities offered will be noticed. Our initial aims were strongly influenced by the template-matching, contextual analysis, facilities necessary to implement an Eliza-type of system [22], [23]. Experience in the design and implementation of contextual editors [8] led to many of the features of the present system. Both the facilities and syntax of string-handling in Basys represent a substantial improvement over the conventional string extensions of Basic that is particularly important in promoting the programming of effective interaction. These facilities have been widely and successfully used by programmers with diverse experience and requirements, and we feel justified in giving some detail here. We hope to illustrate that the necessary power for contextual string analysis can be provided in a natural simplyused form that enables interaction to be programmed as easily as arithmetic.

Our implementation of Basys automatically gave us facilities for manipulating the character strings which form program lines and it was natural to store string data in the same way. Any “program” line beginning with a $ sign is a string variable initially containing whatever follows the $ sign. Such variables are referenced as $<line number> where “line number” is a number, or numeric expression, evaluating to a line number, e.g. :

```
>100 $HI THERE
>PRINT $100
HI THERE
>LET X=99
>P $X+l
HI THERE *
>
```

(Note, the “go-ahead” sign, >, is printed by Basys in edit mode to request input. Hence, in these examples, it serves to distinguish text typed by the programmer from that typed out by Basys).

Apart from its implementation advantage this mechanism for string variables:

1) Enables string constants to be listed as part of the program.

2) Enables string constants to be placed in that part of the program where they are used.

3) Makes string arrays naturally available and, in particular, efficiently implements sparse string arrays.

String expressions containing string constants, variables and literals, and numeric variables converted to strings have a natural syntax similar to that of Basic, e.g.,

```
>PRINT $100 'WHAT IS' X 'TIMES' X
HI THERE WHAT IS 99 TIMES 99
>
```

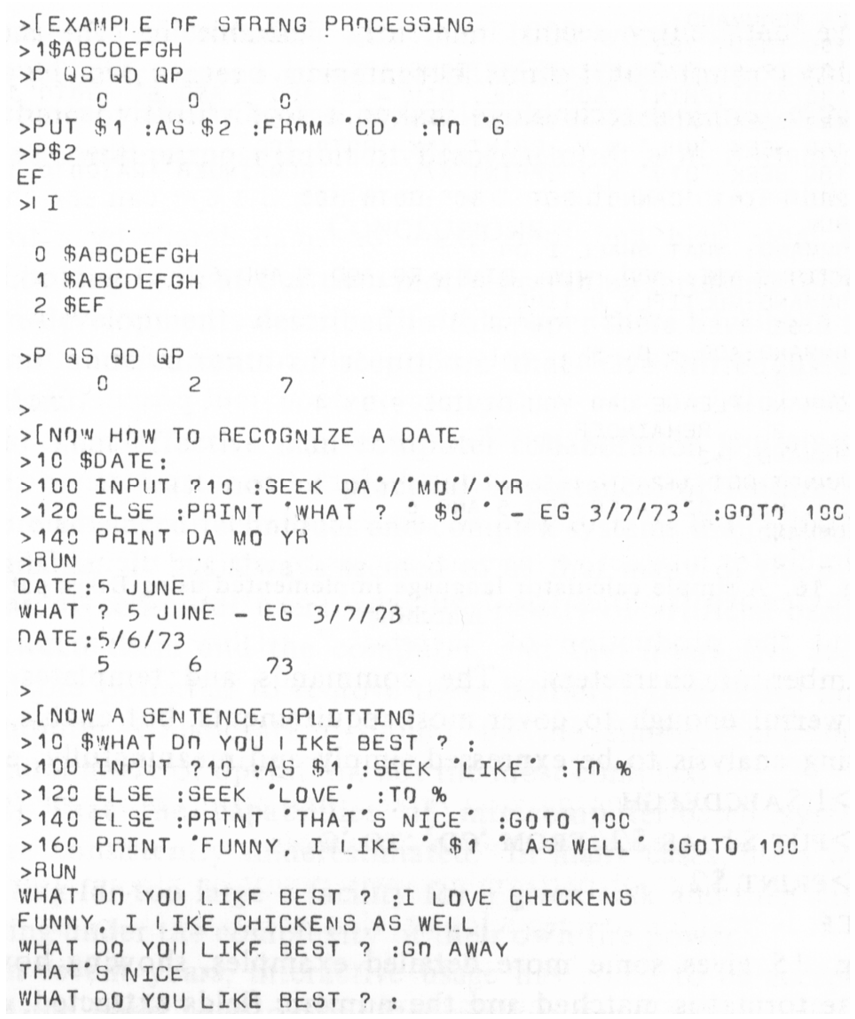
Contextual string analysis is based on the concepts of a source string being analyzed, a destination string to which output may be appended, and. various pattern-matching commands The system variable QS contains the line number of the source string, QD that of the destination string, and QP contains a pointer to a character within $QS. These system variables are automatically set up by the string processing commands but are accessible to the programmer for the more complex string processing.

The command PUT <string expression> sets up a source string by assigning the value of the expression to $QS and setting QP to zero. The command AS <string variable> sets up the string variable as a destination string initially null, and assigns its line number to QD. The command WITH <string expression> is used to append the value of the expression to $QD, typically to replace a pattern matched in the source string.

TO, FROM, and SEEK are template search commands specifying a template to be looked for in the source string $QS starting from the QPth character. If the pattern is found, QP is updated to point to the next character beyond it. If the template does not fit and the match fails then neither QP nor $QD is affected. FROM specifies an anchored search (for an initial template), and SEEK an unanchored search (for an imbedded template). TO specifies an unanchored search in which characters in the source string prior to those matching the template are appended to the destination string. The forms of template include string variables, numeric strings (automatically converted and assigned to a numeric variable), and a specified number of characters. The commands and templates are powerful enough to cover most requirements, but enable the string analysis to be expressed simply and meaningfully, e.g.:

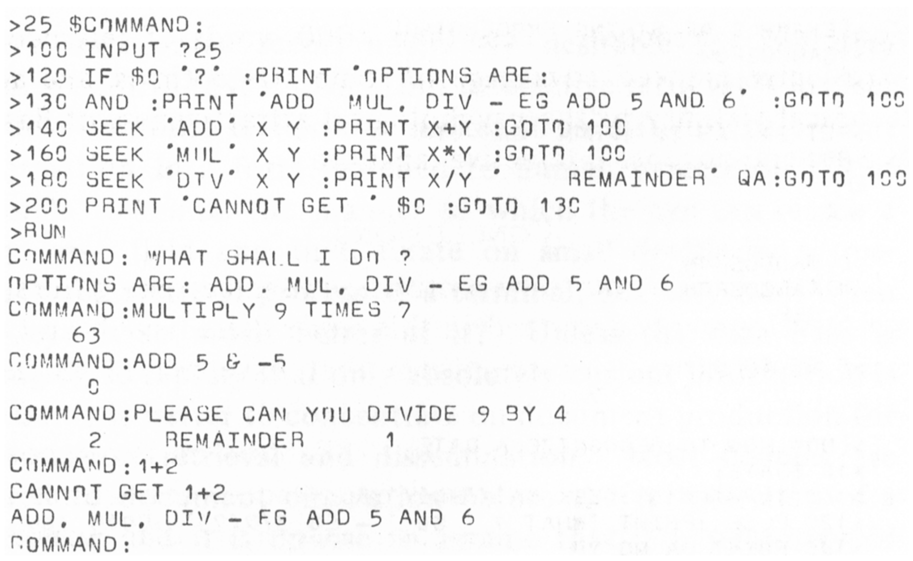
```
>1 $ABCDEFGH
>PUT $1 :AS $2 :FROM 'CD' :TO 'G
>PRINT $2
EF
```

Fig. 15 gives some more detailed examples, showing how a date format is matched and the numeric fields extracted, and how a simple sentence can be split up and reconstructed as a reply. Fig. 16 is a simple "calculator" program with its own syntax that illustrates the use of the string processing to analyze a formal "language."



```
>[EXAMPLE OF STRING PROCESSING
>1$ABCDEFGH
>P QS QD QP
     0     0     0
>PUT $1 :AS $2 :FROM 'CD' :TO 'G
>P$2
EF
>LI

 0 $ABCDEFGH
 1 $ABCDEFGH
 2 $EF

>P QS QD QP
     0      2      7
>
>[NOW HOW TO RECOGNIZE A DATE
>10 $DATE:
>100 INPUT ?10 :SEEK DA'/'MO'/'YR
>120 ELSE :PRINT 'WHAT ? ' $0 ' - EG 3/7/73' :GOTO 100
>140 PRINT DA MO YR
>RUN
DATE:5 JUNE
WHAT ? 5 JUNE - EG 3/7/73
DATE:5/6/73
     5      6     73
>
>[NOW A SENTENCE SPLITTING
>10 $WHAT DO YOU LIKE BEST ? :
>100 INPUT ?10 :AS $1 :SEEK 'LIKE' :TO %
>120 ELSE :SEEK 'LOVE' :TO %
>140 ELSE :PRTNT 'THAT'S NICE' :GOTO 100
>160 PRINT 'FUNNY, I LIKE ' $1 ' AS WELL' :GOTO 100
>RUN
WHAT DO YOU LIKF BEST ? :I LOVE CHICKENS
FUNNY, I LIKE CHICKENS AS WELL
WHAT DO YOU LIKF BEST ? :GO AWAY
THAT'S NICE
WHAT DO YOU LIKE BEST ? :
```

**Fig. 15. Demonstrations of the string-processing commands in Basys.**



```
>25 $COMMAND:
>100 INPUT ?25
>120 IF $0 '?' :PRINT 'OPTIONS ARE:
>130 AND :PRINT 'ADD  MUL, DIV - EG ADD 5 AND 6' :GOTO 100
>140 SEEK 'ADD' X Y :PRINT X+Y :GOTO 100
>160 SEEK 'MUL' X Y :PRINT X*Y :GOTO 100
>180 SEEK 'DTV' X Y :PRINT X/Y '    REMAINDER' QA:GOTO 100
>200 PRINT 'CANNOT GET ' $0 :GOTO 130
>RUN
COMMAND: WHAT SHALL I DO ?
OPTIONS ARE: ADD. MUL, DIV - EG ADD 5 AND 6
COMMAND:MULTIPLY 9 TIMES 7
  63
COMMAND:ADD 5 & -5
   0
COMMAND:PLEASE CAN YOU DIVIDE 9 BY 4
   2     REMAINDER      1
COMMAND:1+2
CANNOT GET 1+2
ADD, MUL, DIV - EG ADD 5 AND 6
COMMAND:
```

**Fig. 16. A simple calculator language implemented using Basys pattern matching.**

These examples demonstrate the operation of the pattern-matching string analysis facilities in Basys in simple situations. Clearly the combined use of all the facilities coupled with computations with, and assignments to, the variables QS, QD, and QP, allows very complex string analysis to be carried out. What should also be apparent, however, is the naturalness of this command family for use in the commonly required string analysis needed to support interactive data entry and conversational dialogues. Our objective was not only to supply string-processing facilities but also to retain the readability, and the transparency of function, which is such an important feature of Basic.

## *C. Variable Scope and Procedures in Basys*

One further area of significant divergence between Basys and Basic is in the scope of variables and the mechanism for passing parameters to procedures. We have found it essential to have variables of local scope and an explicit mechanism for passing parameters by value and by reference. In earlier implementations of Basys, space for all numeric variables was allocated from a single-level store. Once created variable names were static and retained their values between subroutine calls, program overlaying, and so on. Such a simple allocation scheme is adequate for small programs up to about 200 lines, but creates an increasingly onerous memory load on the programmer as they become larger. For the suites of some 30 or more 200-line programs that typified our commercial and medical systems, we found that the use of variable names for different purposes in different places was the greatest single source of errors. In particular, it made modification of large suites very difficult. Wet found ourselves generating elaborate cross-reference packages and losing many of the advantages of rapid development otherwise available in Basys.

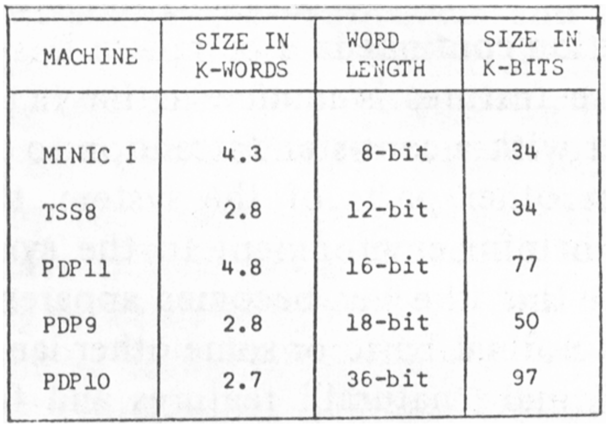
The obvious extension was to restrict the scope of names and this has been done in later implementations by assigning storage for simple variables and arrays from a system stack. This same stack is also used to hold procedure return information and hence new variables created in procedure calls are automatically local. The Basic Gosub command has been dropped as a means of implementing procedure calls. The command DO <line number> in Basys causes execution of the specified line but does not transfer control to it. It is normally used to avoid repetition of long program statements. If, however, the line executed commences with a BEGIN command, the DO automatically becomes a procedure call in which the whole block is executed. Value or reference parameters may be passed to a procedure by listing them after the line number in the DO command, and are picked up by the procedure in local variables listed after the BEGIN command. The command BACK causes a procedure return which clears up the stack and transfers control back to the caller.

Slight variants on these mechanisms are used to implement iterative loops and a block structure. The mechanism outlined has proven both natural and effective. Users do think of a procedure as a separate entity and variables within it as local to it. Again the full impact is felt only when users go back to Basic or an earlier implementation of Basys and have to take their own steps to avoid name conflicts. This problem is not a function of the limited repertoire of names in Basic and Basys since the interactive user naturally uses single character names anyway to minimize the effort of typing.

## *D. Basys Implementation and Extension*

Two practically important features of Basys are its compactness and ease of implementation/ extension. These features are clearly related in that a small program takes less time to implement, and making the logic table-driven for ease of extension also makes for ease of understanding and implementation.

Table I indicates the size of complete core-resident Basys interpreters for various machines. These are comparable representations, although each was interfaced to a different form of device-independent monitor; only the earliest implementation on the TSS 8 is significantly different in its facilities. The sizes stated for the PDP11 appear anomalous but the word-size of this machine is probably better taken as 32 bits rather than 16. Too much should not be read into the table relating to the relative efficiencies of the order codes of the machines, although as a general rule we have found that the smaller the word-size the greater the efficiency of program encoding. The figures given in Table I include code, tables, and workspace, but not user program storage. We generally use a figure of about 20K bits as a reasonable size of user partition for Basys programs of up to about 100 lines complete with their associated variables. This has also proved to be a reasonable size for program overlays in the smaller financial and clinical systems described. The number of machine instructions in a complete Basys interpreter is about 2700. The main source of variation in this figure is the code required to interface input/output, etc., to the monitor of the host system.



| MACHINE | SIZE IN K-WORDS | WORD LENGTH | SIZE IN K-BITS |
|---|---|---|---|
| MINIC I | 4.3 | 8-bit | 34 |
| TSS8 | 2.8 | 12-bit | 34 |
| PDP11 | 4.8 | 16-bit | 77 |
| PDP9 | 2.8 | 18-bit | 50 |
| PDP10 | 2.7 | 36-bit | 97 |

**Table I: Size of Various Basys Implementations**

One would not expect the writing of 2700 instructions to take much time and the implementation of Basys on a new machine is fast and straightforward. First, one writes a family of low-level routines to create a virtual machine with character handling and the required form of arithmetic. Then, one writes a set of intermediate routines that manipulate the Basys data structures. Finally, one writes the top-level routines that evaluate string and arithmetic expressions. The time taken to do this is almost entirely a function of the software development system available for the host machine and is typically about two man-weeks on a machine with a well-integrated disk-based machine-code development system. As we have emphasized previously [8], it is the quality of these developmental resources which has the prime effect on the software development time for well-defined system programs. We have found it worthwhile to write a good assembler and linking loader for a minicomputer before attempting any other software development if its facilities are primitive. Equally we have found it vital to base the development on the best interactive text editor available rather than any other resource. An emulator of one machine on a more powerful, better supported, disk-based configuration is useful, but a good editor is vital. In one recent development it has proved better to edit on a disk-based configuration, transfer for assembly to a magnetic-tape based configuration which has a suitable assembler, and, finally, transfer to the paper-tape based target configuration for testing and debugging. This awkward sequence itself slows down development but is very much faster than using the paper-tape based system alone.

The size of the system is only one of several machine effectiveness parameters. The timings of Basys arithmetic and string-handling are comparable in all implementations, with arithmetic up to 100 times slower than machine code and string-handling up to 10 times slower. For the majority of our applications, these losses are not significant and full advantage may be taken of the interpretive high-level kernel language. Where arithmetic speed is vital, it has been necessary to add small machine code extensions, e.g., a fast Fourier transform or matrix inversion routine. We originally designed Basys as a convenient framework for driving a family of machine-code routines, and it is very simple to add an additional command by putting a new command string in one table, a transfer vector in another table, and a new machine-code routine at the transfer address. Routines in the main package to evaluate arithmetic and string expressions, standard return addresses, and so on, are well documented and accessible to the new routine. Ease of extension of the basic interpreter with fast special-purpose routines has been important in several instrumentation applications otherwise considered unsuited to minicomputers programmed in high-level languages.

The interpretive execution of Basys is, however, its greatest limitation in more general applications. Several of our users now have large numeric and nonnumeric data analysis packages in Basys that would be far more widely used if they were faster. The advent of customer-microprogrammable minicomputers from several manufacturers has led us to investigate the possibility of putting a substantial part of Basys's 2700 instructions into microcode. Less than 1000 words of microcode would give a speed advantage of some 10-15 times, and add technical viability to the psychological attractiveness of Basys as a general-purpose interactive language.

## CONCLUSIONS

Looking back at our own motivation in undertaking the system developments described in this paper there have been two main undercurrents of scepticism that have stirred us into action.

1) That effective man-computer collaboration is not possible, or at least not of practical importance, or requires far more advanced techniques and complex systems than currently available. It has always seemed to us that many of the problems of computer users were the results of artificial barriers between them and the computer, and that interactive access had the potential to remove these barriers. Interactive man-computer collaboration should be the simplest and most natural mode of operation, not the most complex.

2) That the capabilities of minicomputer-based systems were consistently underestimated. In many cases, users were getting far too large a facility for a given task and then suffocating under the complexity of their own fire power.

In recent years, interactive usage has come to be accepted and the range of minicomputer applications greatly expanded. However, a degree of scepticism is still widespread and it is clear that interaction alone does not necessarily mean effective collaboration, neither does the way in which minicomputers are used necessarily take full advantage of their potential as small flexible system modules. We hope that this paper provides not only further detailed evidence of the practical potential of interactive minicomputer-based systems, but also some guidelines as to their development and use, together with examples of facilities and techniques against which other systems may be evaluated. In particular we have shown that a high-level language on a minicomputer can be very compact and does not detract from the attractive small-size low-cost aspects of minicomputers. We have shown that contextual string-handling can also be provided compactly in a simple and natural form as readily used as arithmetic. Given these facilities, we have shown that interaction itself can be programmed according to simple rules which aid the non-computer-orientated user and avoid his having to adapt to the peculiarities often present in computer systems.

## Acknowledgements

The systems described have been the work of many diverse groups in which we have been involved largely as technical system designers and implementors. Many others must take credit for both imagination and courage in envisaging how such systems might be applied and actually putting their own efforts and credibility behind the applications. In particular, we owe a large debt to our colleague Dr. J. Gedye who has been the foremost proponent of interactive computers in medical applications in Britain. He has been responsible for generating interest in many of the applications described and has played major roles in their design and application. It would be invidious to single out particular persons in the many commercial organizations with whom we have collaborated. To them all we are grateful for the opportunity to work together on actual problems and test out the techniques in an atmosphere of harsh reality. Finally, we are grateful for the enlightened attitude of our institution in encouraging both the interdisciplinary teams and the external collaboration necessary to such systems research.

## References

1. B. R. Gaines, P. V. Facey, and J. Sams, "An on-line fixed interest investment analysis and dealing system" in *Proc. Eur. Computing Congress*(EUROCOMP74),pp. 155-169, May 1974.
2. T. C. S. Kennedy and P. V. Facey, "Experience with a minicomputer-based hospital administration system," *Int J. Man-Machine Studies,* vol. 5, pp. 237-266, Apr. 1973.
3. J. L. Gedye, "The use of an interactive computer terminal to assess the effect of cyclandelate on the mental ability of geriatric patients," in *Assessment in Cerebrovascular Insufficiency.* G. Stocker *et al.,* Eds, Stuttgart, Germany: Thieme, 1970, pp. 29-42.
4. —, "The use of an interactive computer terminal to simulate decision-making situations," in *Artificial and Human Thinking* A. Elithorn and D. Jones Eds. Amsterdam, The Netherlands: Elsevier, 1973, pp. 102-124.
5. —, "A research methodology applicable to the study of the therapeutic effects of meclofenoxate in elderly patients," to appear in 1975.

6. R. H. Atkin, “From cohomology in physics to q-connectivity in social science,” *Int J. Man-Machine Studies*, vol. 4, pp. 139-167, Apr. 1972.

7. —, “Multi-dimensional structure in the game of chess,” *Int. J. Man-Machine Studies,* vol. 4, pp. 341-362, July 1972.

8. P. V. Facey and B. R. Gaines, “Real-time system design under an emulator embedded in a high-level language,” *Proc. Brit. Comput. Soc.,* DATFAIR 73, pp. 285-291, Apr. 1973.

9. D. F. Parkhill, roe *Challenge of the Computer Utility.* Reading, Mass.: Addison-Wesley, 1966.

10. W. D. Orr, Ed., *Conversational Computers.* New York: Wiley, 1968.

11. J. C. Shaw, “Joss: a designer’s view of an experimental on-line computing system,” in *Rec. AFIPS Fall Joint Comp. Conf., vol.* 26, Pt. 1. Baltimore, Md.: Spartan, 1964, pp. 455-464.

12. J. W. Smith, “Joss II: design philosophy,” in *Annual Review in Automatic Programming,* M. 1. Halpern and C. J. Shaw, Eds. Oxford, England: Pergamon, 1971, pp. 183-256.

13. J. G. Kemeny and T. E. Kurtz, *Basic Programming.* New York: Wiley, 1967.

14. , *Basic* (Users’ Manual). Hanover, N.H.: Dartmouth Publ., 1968.

15. J. L. Gedye and E. Miller, “The automation of psychological assessment,” *Int J. Man-Machine Studies*, vol. 1, pp. 237-262, 1969.

16. —, “Developments in automated testing systems,” in *The Psychological Assessment of Mental and Physical Handicaps,* P. Miller, Ed. London, England: Methuen, 1970, pp. 735-760.

17. F. Goldman-Eisler, “Hesitancy and information in speech,” in *Information Theory. C.* Cherry, Ed. London, England: Butterworths, 1961, pp. 162-173.

18. J. McCarthy, D. Brian, G. Feldman, and J. Allen, “Thor—a display based time-sharing system,” *AFIPS Spring Joint Comp. Conf.,* vol. 30. Washington, D.C.: Thompson, Apr. 1967, pp. 623-633.

19. A. van de Goor, C. G. Bell, and D. A. Whitcraft, “Design and behavior of TSS/8: a PDP8 based time-sharing system,” *IEEE Trans. Comput,* vol. C-18, pp. 1038-1043, Nov. 1969.

20. *TSS/8 MonitorManual* (DEC-T8-MRFA-D). Cambridge, Mass.: Digital Equipment Corp.

21. *TELCOMP 2 Users’ Manual.* Cambridge, Mass.: Bolt, Beranek, and Newman.

22. J. Weizenbaum, “Eliza—A computer program for the study of natural language communication between man and machine,” *Common. Ass. Comput Mach.,* vol. 9, pp. 36-45, Jan. 1966.

23. J. Weizenbaum, “Contextual understanding by computers “ *Common. Ass. Comput Mach., vol.* 10, pp. 474-480, Aug. 1967.

24. R. E. Griswold, *The Macro Implementation of Snobol 4.* San Francisco, Calif.: Freeman, 1972.

25. J. F. Gimpel, “A theory of discrete patterns and their implementation in Snobol 4,” *Common. Ass. Comput Mach.,* vol. 16, pp. 91-100, Feb. 1973.

26. B. R. Gaines and P. V. Facey, “Basys: a language for programming interaction,” to be published in *Int J. Man-Machine Studies,* 1975.

27. *QUASIC User Manual.* London, England: Questel Ltd., 1970.

28. *QUASAC UserManual.* London, England: Questel Ltd., 1971.

29. *AIMS UserManual.* London, England: Arbat Consultants, 1973.

30. *MINSYS User Manual.* London, England: Micro Computer Systems, 1974.

31. *Basys User Manual,* Dep. Elec. Eng. Sci., Univ. Essex, Colchester, England, 1974.

32. Southend-on-Sea Hospital Management Committee, Experimental Computer Project System Study Report, Rep. 324, Mar. 1972.

33. Southend-on-Sea Hospital Management Committee, Experimental Computer Project Waiting List System—Interim Evaluation, Rep. 377, Dec. 1973.

34. Datasaab-Medela Technical Description, Datasaab 8132E 60.00.56 Saab Aktiebolag, Linkoping, Sweden.

35. I. Brolin, “A computer-based terminal for radiological reporting,” *Int J. Man-Machine Studies,* vol. 1, pp. 211-235, Apr. 1969.

36. J. Martin, *The Design of Man-Computer Dialogues.* Englewood Cliffs, NJ.: Prentice-Hall,1973.

37. T. C. S. Kennedy and R. Edmunds, “An examination of training problems with naive computer users,” in *Proc. Eur. Computing Congress* (EUROCOMP 74), pp. 411-425, May 1974.

38. T. C. S. Kennedy, “The design of interactive procedures for manmachine communication,” *Int J. Man-Machine Studies,* vol. 6 pp. 309-334, May 1974.

39. S. Treu, “Interactive command language design based on required mental work,” *Int J. Man-Machine Studies,* 1975, to appear.

40. B. R. Gaines, “The role of randomness in cybernetic systems,” in *Proc. Conf Recent Topics in Cybernetics.* London, England: Cybernetics Soc., Aug. 1974.