chase order. Each shipment is for a specified quantity of parts to be shipped on one date and delivered on another date. The status of delivery is recorded by a code.

5. *Relation name: actual receipt*
A quantity of parts for an actual shipment made on one day is received on another day. This quantity may not be the same as that promised.

6. *Relation name: reject*
Quality control require that some parts be inspected. The quantity of parts which are rejected are recorded on a reject report together with the date of shipment, the part number and the purchase order number. A unique number is allocated to each reject report and a separate report is made out at the end of each working day for each shipment.

7. *Relation name: buyer*
Each buyer is allocated a code number. His name is recorded as well as the department that employs him.

8. *Relation name: supplier*
A supplier may have many addresses. A unique number is allocated to each address. A supplier's name is recorded with each address as well as street, city, state and zipcode.

9. *Relation name: payment committed*
A buyer is authorised to make commitments of money, to an agreed level, with particular suppliers. The amount of money committed by each buyer is recorded by quarterly and annual totals for each supplier. Payments to the supplier are also recorded in quarterly and annual totals for each buyer.

10. *Relation name: PAYMENT*
Payments are made against invoices received from suppliers. These invoices may refer to many purchase orders. The date of each payment is recorded together with the date of payment expected by the supplier, the gross amount payable and the net amount paid. The status of payment is indicated by a code.

*CONCEPTS*

| | |
|---|---|
| BUYER | AGREEMENT |
| PURCHASED | PAYMENT TERMS |
| IDENTIFIED | MONEY |
| CODE | TIME INTERVAL |
| PLACES | PART(S) |
| PURCHASE ORDER | DESCRIBED |
| AUTHORISATION | TEXT |
| MANAGER | ALLOCATED |
| ATTENTION | REFERRED |
| PERSON | EMPLOYED |
| NUMBER | UNIT OF MEASUREMENT |
| PLACED | TOTAL QUANTITY |
| SUPPLIER | AVERAGE PRICE |
| DATE | ACTUAL SHIPMENT |
| DEPARTMENT | RECEIVED |
| USES | INSPECTED |
| LOCATION | REJECTED |
| | RECORDED |

**Conclusion**
The authors believe that greater efforts must be made to ensure that the user has a fuller understanding of the implications of the design produced by the analyst. Towards this end this paper has listed two complementary approaches. Firstly, the business rules which specify the relationships between relations and secondly the narrative text which describes each relation.

Both techniques allow the user to understand more fully the implications of a design. In consequence the user's approval of the outline design will be based on a clearer understanding. This should lead to fewer changes being made later due to errors in the design, with a resulting decline in development times and costs for new projects.

**Acknowledgement**
The authors wish to acknowledge the contribution of Ms. G. Butler of Exxon Corporation in the introduction of the business rules concept and the case study used in the logical data base design course. They also wish to thank Exxon Corporation for permission to publish the paper.

**References**
CODD, E. F. (1970). A relational model for large shared databanks, *CACM*, Vol. 13 No. 6, pp. 377-387.
WIEDERHOLD, G. (1977). *Database Design*, McGraw-Hill.

# The user interface

E. B. James

*Computer Centre, Imperial College, Exhibition Road, London SW7 2BX*

**Why we need a new approach**
Our purpose here is to discuss new approaches to systems analysis and design which may help to alleviate a growing dissatisfaction with the performance of existing computing systems. These systems are under attack on two main charges. The first suggests that the systems do not do what is required by the users; there is a mismatch between the views of the designers and the users as to what was originally required of the system. The second charge is that it is far too difficult to make the computing system do anything useful. Other papers in this symposium concentrate on the first issue. We propose to concentrate on the second issue, which is concerned with the quality of the 'user interface' as it is called. We feel that this is particularly urgent because the type of user typical of a computing system is changing rapidly. The far less experienced user who will soon make up the principal proportion of all users will have, it appears, even greater dissatisfaction with existing computing systems and their dissatisfaction is likely to become rapidly more vocal.

In this discussion, we aim to define those qualities which contribute to ease of use and which seem to be missing from existing designs. We try to determine who is capable of defining these qualities in detail and we look at some previous attempts to satisfy these requirements. Then we consider the current situation in the hardware of computing systems, where it seems that the rapid changes could assist a better type of user interface design. We consider how we may exploit this new situation and we propose techniques to achieve this. Finally, we describe a project which has adopted the new approach and which may encourage other designers.

**The new user**

Let us consider the new type of user, who may make obsolete existing methods of interface design. In the early days of computing systems, the users would have been difficult to separate out from the designer of the computing system. Certainly they were technologists closely related and having ready access to the original systems designers both on the hardware and software sides. Now, in increasing proportion, the users will be people without any direct experience of computer design. They will be 'the person in the street'. In earlier days, the main use of computers was to ease the burden of calculation. Now, while this activity is still useful, the computer is seen as providing a far wider range of support activities. Earlier, the specialists who used the machines were prepared to expend several months or possibly years of hard work in preparing a working program, in order to obtain results which would have taken many lifetimes to prepare by hand. Now, the new users require a range of comparatively simple services from the machine and are certainly not prepared to expend an exorbitant amount of intellectual effort in obtaining these straightforward results. This would seem to be particularly the case when the machine does not produce the expected results. The experienced professional user has in the past shown an enormous tolerance to the unexpected behaviour of a system and has even obtained a great deal of pleasure in the detective work necessary to find the reasons for its incorrect operation and to correct the program. The new user will have no interest in helping the computer out if it goes wrong.

**Defects in the system**

What are the drawbacks of computing systems as seen by the inexperienced user? We suggest that the first characteristic is unreliability: the machines do not regularly and adequately deliver the goods. They require continuous prompting sometimes to produce anything at all. The second characteristic is intolerance: the computer system requires an absolutely correct stream of directives to carry out its functions. If a tiny slip is made in this specification, then nothing seems to work. A third drawback is that computers are impersonal: that is, it seems to require a very wide general knowledge of *all* the things that the computer can do before it is possible to instruct it correctly to do the *particular* thing which the user wishes. Who then is able to define the required characteristics of a better system?

The natural suggestion would be that the users themselves should define carefully what they require of a system. In practice, this raises all sorts of problems. If a particular user is selected to represent the users in general, say in consultations with systems designers, then there is the danger that this person will be a symbolic representative who is in practice unable to talk for the other users, nor even possibly to communicate with the existing experts at the level of discussion which is at present unfortunately relevant to specifying the needs. The presence of user representatives on a formalised committee to which systems proposals are presented may be ineffective, since the actual decisions taken by the systems people are clearly not made at those meetings, and the whole operation may well smack of paternalism. In any case we must have some means of getting over the well known impasse where the designer says 'What do you require?' and the user says 'What can you provide?'—it is rarely possible to see the flaws in a design from the user's viewpoint until the system has been in action for some time. Therefore the sensible approach would seem to be to present a trial system and then to provide all users with ample opportunity to state their views on it frankly and constructively.

At this stage it may well be pointed out that there are many groups of designers at work who believe that they are genuinely working towards a better situation for users. We suggest that many of the current attempts are misguided for various reasons. Some of these attempts involve the construction of non-solutions to the *real* problem. This occurs when a designer, however well intentioned, designs a system effectively for *personal* use rather than the use of a large number of people who do not have the experience of the designer. As Bickley (1966) has pointed out, this provision of pseudo-solutions takes place at a much deeper level in other formal sciences. He suggests that a formal mathematical proof is really laid out for the satisfaction of the person who thought it up in the first place and to the outsider gives no hint of the way in which the theorem to be proved came to be conceived and investigated, which is the really significant point. Surely no effective user interface can be provided if every user is assumed to have the same knowledge as the *designer* of the system?

Other workers it seems are producing solutions to some problem other than that of the user. For instance the activities of those trying to prove programs 'correct'. De Millo (1979) discusses this strange situation at length and indicates how unhelpful some of these activities are. Again, many experts are applying themselves to the solution of problems connected with an earlier situation in practical computing machines. For instance, they are involved in the design of large complex languages which presupposes the construction of monolithic programs to do a clearly defined single task. Such is simply not the case in present day programming, as Winograd (1979) points out.

Finally, another group of well meaning designers are working on the solution to subproblems which, it turns out, do not satisfy the *overall* conditions of the total problem. For example, a multitude of very powerful packages are being produced, where it is necessary for the user to memorise the meanings of perhaps ten different parameters in say twenty different commands before being able to specify what is required. If this is repeated in a *total* computing system it leads to an intolerable load on the user, particularly if that package is not going to be used at all frequently.

**The changes in the machine**

Let us now consider the present situation in hardware availability. This is now in a state of flux, due to the advent of the microprocessor, but certain new factors have clearly come to stay and they provide some very important new opportunities to improve the user interface. Let us consider the old situation compared with the new. The older type of system involves the construction of a monolithic program to do a well defined job. The new situation seems to require a wide range of facilities which can be joined together easily by the user to achieve the particular task.

The old system involved no genuine interaction with an executing program. The new system presupposes a continuous interaction with *every* process. The old type of systems approach places a great stress on obtaining the maximum performance from inevitably limited hardware. In the new situation, with the hardware dramatically cheaper, it is often convenient to simplify software design problems by multiplying up on the hardware side, and this will be the overall economic answer. Finally, in the old situation we continually see the emphasis on the *program* as the significant feature of the design. In the new situation, the programs are no longer of such great significance and there is much more emphasis on the *data* as the principal 'given'. So how may we plan to exploit this new situation? First, we must make the hardware work much harder than before. In particular we may dedicate a considerable amount of hardware power to looking after the interface between user and the rest of the machine. Secondly, we can promote genuine interaction so that the user does not have to wait long between opportunities to redirect a process which is not going as expected. Thirdly, in the same spirit, we should make it simple for the user to experiment with various ways of processing data in order to determine more effectively what is actually required. Finally, we should provide the processing programs as a number of simple modules which are designed for easy interconnection. In particular, it should not be necessary to do awkward conversions of data format between each section of program.

**Interface qualities**

We looked earlier at desirable qualities of the user interface, which have been conspicuous by their absence. Newman (1978) provides a valuable summary. Let us concentrate on the psychological aspects of certain of them which seem particularly significant. The first and most important quality of the interface is reliability. When everything is operating smoothly, this is mainly a question of providing a consistent response to user activities. However, reliability becomes much more obvious when things are not going right. A first range of problems will be encountered when there is a degradation in the computer system. It is essential for a continuous and consistent response to be maintained as far as possible even when parts of the system are not working. Another range of problems arises when the user performs incorrect operations or makes incorrect requests. It is essential in this situation that the system continues to perform in a consistent manner and, most important of all, continues to communicate. Clearly this implies a great deal of tolerance to the user's behaviour on the part of a computing system.

The second quality is adaptability. We are concerned here particularly with the ability of the computer system to provide a dialogue which is consistent with the user's previous experience. This implies that the system is able to learn a 'required' pattern of behaviour from a particular user and also to remember and reintroduce this pattern when the same user is in touch again. There is no way in which a system with a permanently fixed 'level' of response can satisfy a group of users with a very wide range of experience. The third quality is self-sufficiency. This is an aspect of reliability, in that the user is not required to go *elsewhere* for help in solving a problem. This implies in turn that the system is able to provide training in its own use and therefore to act as a computer assisted learning system whenever required.

A fourth quality, ease of use, is connected very much with reliability and tolerance. Clearly, 'easy' is closely related to the personal background of the user. This implies that the system should be as simple as possible but also that when a difficulty is encountered, a fullscale teaching system should be brought into action. In every case there must be no doubt in the mind of the user as to what is required next. 'Ease of use' also implies that the system adapts to the human method of problem solving. This is likely to be reflected in an overall hierarchical organisation of activities (Weiss, 1971). At the same time, however, each activity must accept incremental additions at any point. Ease of use also implies efficiency but it cannot be stressed too strongly that this is not the efficiency in the mind of the typical systems designer, who is concerned with the efficient use of machine resources. By efficiency here, we mean a minimum use of the *user's* resources in obtaining the required 'results' from a computing system.

We believe that the qualities just described are absolutely fundamental to a good interface design; and yet, our experience of many different computing systems suggests that these objectives have *never* been met in practice

What design techniques will help us to achieve the desired qualities? Some basic principles already familiar in computer assisted learning are very relevant here. The first one is that the principal objective of any output from the computer is to inform the user what is required next. Never at any stage of the communication process should the user be unsure of what is required. The second principle concerns the need for a user to suspend what is being done at any time, and to be able to start again from that point. It should be possible to remind the user of what had happened previously. A final principle concerns the wide variation in different users' experience. An experienced user's time must not be wasted in going through tedious question/answer sessions. This implies different levels of communication varying between the very educational and the very concise which may be chosen by the user or, one day, determined dynamically by the system as it converses. Naturally, the system must show a great deal of tolerance to trivial mistakes in format or spelling (James and Partridge, 1976). It is comparatively easy to define qualities and determine principles. Many designers have been dissatisfied with an existing system and have set off to define a radically new and better one from scratch. Eventually we must learn from Popper (1973) that such attempts at changing the world are foredoomed to failure. The principal objective must be to make things rather easier for large numbers of users on *existing* systems and so we must accept the inherent constraints.

**Protective ware**

Our solution to this problem is to insulate the user from the existing hardware and software by means of a new sort of *protective* ware. For example, concerning communication between the machine and the user, we never allow the user to see any peculiar system messages from the operating system of a particular manufacturer. All systems messages are intercepted by the protective ware, which is a specially designed program controlling the interface. This program then plans what to do. If necessary, the user is told of a particular required action in words which are meaningful to him, rather than to a systems specialist. In many cases, it is not necessary to tell the users anything. Conversely, when the users wish to communicate with the machine, they are never required to use the particular operating systems jargon. The protective ware interacts with the user and generates the necessary system commands. A separate microprocessor is obviously very suitable for use in this intermediary role. As these methods develop more fully we may look forward to the abandonment of formal programming languages by the majority of users. But, heeding Popper, we must attempt to alleviate the *immediate* situation first.

Our current work is directed towards integrating the same simple principles into the normal programming users' environment. In this case, it will be necessary to interpret a wide range of users' requests and therefore to provide a protective coat for a considerable portion of the job control language which requires to be activated in order to carry out what the user requires. We have started by writing an interactive interpreter for basic file processing operations. This runs in a local, dedicated microcomputing system and generates the protocol required to process the users' files on the main machine. A timesharing link to the main machine is automatically activated and the generated protocol is initiated, resulting in the required file processing on the main system, with the transfer of necessary results down the line to the user. As a concrete example, the request from the user: 'print' results in a query as to whether the output is required on line printer, microfilm or microfiche and, based on the reply, the necessary system commands are sent downline to the main system. Those who have struggled with the totally different and complex series of commands necessary to realise these, from the users' point of view, similar requirements will be grateful for the assistance we provide. All the systems commands we have not protectively covered can be requested in the original format. The net result is that every user can gain *something* from using our system.

What has this to do with a new approach to systems analysis and design? We suggest that further developments in sympathetic, easy to use systems will radically alter the role of the systems analyst and designer in the future. No longer will the main effort be concentrated on getting the processing operations to work on a complicated and intolerant system. Instead, all the 'human engineering' aspects can be concentrated on to the greater satisfaction of users. Perhaps the systems designer can be entirely dispensed with and the user can be allowed to approach the friendly system directly. The systems analyst's role will then be mainly to advise and encourage in the initial stages of deciding what is required.

This seems to be in stark contrast to the conventional image of the analyst as primarily an expert on the 'system'. Maybe we will require an entirely different sort of person, trained in an entirely different way (Podger, 1978). But that is another problem.

**References**


BICKLEY, W. G. (1966). Some thoughts on Mathematical Thinking, *Mathematical Gazette*, Vol. 50, No. 371, pp. 1-8.

DE MILLO, R. A., LIPTON, R. J. and PERLIS, A. J. (1979). Social processes and proofs of theorems and programs, *CACM*, Vol. 22, No. 5, pp. 271-280.

JAMES, E. B. and PARTRIDGE, D. P. (1976). Tolerance to inaccuracy in computer programs, *The Computer Journal*, Vol. 19, No. 3, pp. 207-212.

NEWMAN, I. A. (1978). Personalised user interfaces to computer systems, *Proceedings Eurocomp 1978*, Online conferences.

PODGER, D. N. (1978). *High level languages—A basis for participative design*, Manchester Business School.

POPPER, K. (1973). For example see p. 85 of Magee B. *Popper* Fontana/Collins.

WEISS, P. A. (1971). *Hierarchichally organised systems in theory and practice*, Hafner Publishing Co, New York.

WINOGRAD, T. (1979). Beyond programming languages, *CACM*, Vol. 22, No. 7, pp. 391-401.


# Discussion on 'New approaches to systems analysis and design'

**Morning discussion**

*Keith Watts* (Thames Water Authority)

I see the papers we have had moving in two directions and I can see a conflict. I'd be grateful if it could be resolved for me. In Peter Hammersley's pivotal talk he said that he saw systems analysis inevitably involving data base in future, and before that Enid Mumford had shown us that in her analysis of users' requirements the requirement for autonomy figured fairly prominently. Peter Prowse said you must have central control of the data. I see data base development as really demolishing the users' autonomy in many ways because, well, Tom Gilb has said that data base technology is the last round of the computer professional trying to exert control; I don't see it as that but I do see it as inherently centralist. I have this problem at the moment to decide data base or not in a certain range of application areas and giving more priority as I have done to the social aspects of the introduction of change my view has been that data base should be avoided unless it becomes inevitable because of the invasion of autonomy. It is not so much the autonomy of the clerical user as the autonomy of the manager, the manager who feels very much threatened that his autonomy is being taken away and he is giving his data to a centrally controlled data base.

*Peter Prowse*

The way I see it is that the user does not need to lose his authority over data. Now I talked about a logical model which consists of hundreds, maybe thousands, of relations. They are aggregated together into a data base. But we find that individual relations (they might be implemented by records or segments of data)—individual relations are generally updated by a particular part of the business. You then have to be very careful because if you put all data into a hierarchy then you have to have only one group that controls the root of the hierarchy.

That is, generally, common things which are common to the corporation. Now as you go down the hierarchy of data in the subject there are many different sorts of data and each little aggregation is controlled by a different user group. So they still feel they can update and control their own data—but the organisation of it is done consistently and they cannot alter the little bit at the apex. But most people are willing to give up something like that in order to get the benefits they see from a data base approach.

*Peter Hammersley*

I think, at least I hope, what I said was that if you are going to have a totally integrated system then I would support the data base approach. In the system which we implemented we went for the data base approach and nothing that happened subsequently convinced me that the approach was wrong. But that is if you are going for an integrated system. Now in that final postscript that I gave I pointed out what I see as being the problem of the future, and possibly the problem to which you are addressing yourself, and that is that the user will not wish to have a corporate central data facility, that he will go out and he will buy his own micro with turnkey software, and he will build his own system and he will keep his own data, and the problem for the analyst who feels that a central data base system is desirable is whether or not developments of that kind are capable of being controlled. Now I am afraid I have not answered the question—I have only posed it because only time will answer it, and we are talking about something that is going to hit us in two or three years' time. I do not know if there is anyone here who has actually hit it already? Maybe if there is we ought to be hearing about it. But this is a problem that is going to hit us very shortly and therefore I hope when I spoke I actually presented the conflict myself by saying, 'This is what we did, but it may not be possible for us to do it the next time round for all sorts of reasons'. I am sorry I cannot answer your question except to agree with you that it is a real problem.

*Frank Land*

May I make just one comment. I think one of the very important elements in the introduction of any system is the process by which it is introduced and if a centralised data base, for example, is introduced by an authority who says 'that is how you are going to do it' then I think you have got much less chance of success than if it springs from the desires and requirements of the people who are involved in the process, and who are ultimately going to be the users of the data base. As far as the technology is concerned, of course the idea of distributed data bases, so you can distribute your data base to the nodes which use them, is also becoming a possibility but it seems to me the most important element is that we have got to have a process of analysis, design and construction which really does get the approval of those who are going to be the end users. If you do not have that then what Peter said is going to happen: you are going to