One man's meat:
Part I—The uses of adversity

By F. I. Musk *

This is the first of a series of papers which attempt to define one computer user's philosophy This philosophy will be valid only if the user can prove he has striven to adhere to it. The necessary proof will be furnished by his existing software. Each software description will be a paper in its own right.

They would sit at breakfast in Forte's, watching the sun rise above the M1 on a summer's morning. They were teams of two, part of a shuttle service between Coventry and London. We sent them off in pairs every four hours. They came back in groups like buses, boisterous groups of six and eight. If we lost any, they turned up again later. The party seemed to be going on for ever, for our computer was aching in every dry joint. Our computer policy then was to avoid coming to blows with our customers. This is what I mean when I say that any attempt by an industrial user to adhere to a fixed computer policy is likely to fail.

We had some interesting programs on. We were running a planning and scheduling scheme linked to stock control, invoicing and sales ledger. We had written a linear-programming routine, and could design distillation columns and heat exchangers. We had built up a colour matching catalogue for standard shades. We did multi-factor analysis. We wrote our own sort routines. We had a forward looking policy, we felt—and then this. We all but succumbed.

From our defensive posture, we crept out and embraced the precept of reliability. We would in future avoid discs, drums and other mechanical devices, which had given us such rich experience of involved breakdown and inscrutable error. Our software would be venturous in effect, but modest in structure. There would be no sophisticates in our team, for good ideas were simple ideas.

This was our new policy, and it served us well. Now that we are leaving our third machine, and again stepping into the unknown (perhaps the abyss), we ought to see if there is some immutable principle which served us in the past, and which we can take into the future. If there is, it cannot be a policy. Our past is strewn with broken policies. We would not like to think it was brute urge for survival, although at times we have been very close to that. Perhaps it is a philosophy. If it is, it should be reflected in the software we have developed. These articles will be an attempt to define such a philosophy, tenuous then, becoming clearer now, and illustrated by our software.

As each piece of software is introduced, it will be accompanied by a detailed description of its function and its anatomy by its author. Our philosophy will not be applicable, may not even be of interest, to many computer users. Neither may our software be applicable, but it will be of interest. Let us start by assuming that what follows is (and was) the first tenet of our philosophy.

> "To an industrial concern, the initial function of computing is to provide management at all levels with all relevant but no other data, in the most easily assimilated form, at the precise moment when a policy decision has to be taken."

This should not be mistaken as the only such tenet, and not even perhaps the most important. It must, however, be the first because some movement towards its implementation is a necessary prior condition for further development.

Developing a data base

A number of years ago, a colleague† and I were asked to look at possibilities for operational research in a new factory. Our investigations showed not only that insufficient background data were available for analysis but that the existing record system would rarely produce the clues necessary to develop relationships of relevance to a systematic study. Where appropriate records did exist, the data became buried in complex special purpose end forms and summary statements. Since a punched-card installation was available, one of our minor suggestions was that data might be held on cards in an uncombined way, each basic variable having its own field. This pack of cards would constitute a Data Bank, as we then called it, from which information could be culled to provide not only routine but also *ad hoc* tabulations. This would have made future operational research (or any other) investigations easy. We were ignorant at that time of the crudity of data processing by "conventional" equipments, and wiser counsels (or apathy) prevailed.

Now, however, with the emergence of data processing on the computer scale, such a proposition would have

† The late Dr. E. D. Totman.

* *Computer Dept., Courtaulds Ltd., Matlock Road, Coventry.*

something to commend it, and perhaps it is fortunate that so many organizations in this country start computing for this purpose. Even organizations which have a long tradition in industrial statistics or operational research tend to introduce computers on the simple basis that by so doing they will reduce clerical cost.

Reports by working parties may contain closely reasoned arguments on the supply of accurate and up-to-date information for management control. They may dwell upon the possibilities of reducing process costs by the use of optimum methods in planning and scheduling. They may suggest that routine operating data, now used only for day-to-day running, can be saved and sifted for large-scale analysis, and that this may complement research effort in other directions. They may plead the usefulness of computer models, most particularly to demonstrate the feasibility of on-line working. It is, however, most difficult to produce a concrete example within the organization, and in the absence of a computer, which will prove the cost effectiveness of such applications adequately to justify on that basis the installation of a computer. No argument is so telling as a costing.

On the other hand, it is possible to set down side by side for comparison the costs (or close estimates) of existing clerical methods against the cost of a computer method. This being the case, all initial efforts must be directed to transforming the clerical method to a computer method. If no insuperable problems appear in programming, if the computer can be induced to take its load in the computer time available, if deadlines can be met and the equipment is reliable, and if the costings were close enough, then success is assured.

At this stage, an ideal loading would consist of customers' orders, despatches, sales invoices, sales ledger, cash reconciliation, sales statistics, on the one hand, and requisitions, purchase orders, receipt of goods, allocation of invoices, purchase ledger, payment of accounts, and stores issues on the other. If, then, payrolls for wages and salaries are added, with bonus calculations where these apply, the system implicitly includes job and materials costing, and a push can be made towards a complete costing system.

A likely computer loading at this stage would not include all of this, but it would include sufficient elements of this *data base* to plan movement forward to the next stage.

Exploiting the data base

Let us suppose we are equipped with a no doubt efficient, marginally profitable, utterly pedestrian accountancy system on our computer; how can we best aspire to the first tenet of our philosophy? This requires the development of more than simply the routine tabulations which are usually found in such a system. These routine tabulations occur periodically, sometimes cumulative, sometimes as comparisons with the previous period, or with the same period last year. They tend to occur in multiple copies, the original reasons for which are historic. It was easier in the old punched-card machinery days, or clerical days, to provide many copies of a complex document, each recipient being interested in only one fragment, not entirely happy with his return, but knowing that only by mountainous labour could the return be changed to the slightest degree. The aim is to provide to anyone on request (or within 24 hours) a specific return tailored to the enquirer's precise need. If this can be done, the mountainous piles of output may be avoided. Ideally, there would be no routine reports at all. There would be many small one-copy random returns. Since we are never to be caught by surprise, we cannot resort to ordinary programming methods. Even (for this is commercial stuff) COBOL will be insufficient for this purpose.

We must design a program which will perform the following tasks:

1. Extract and store a function of the contents of a field according to given criteria.
   These criteria can be one or more attributes of the contents, or an attribute of a neighbouring field or fields in the same item or record, or combinations of these.
2. Arithmetically combine a function of a field with a function of a field, where both fields are in the same file.
3. Arithmetically combine a function of a field in one file with a function of a field in another file.
4. Sort resultant fields according to a given hierarchy.
5. By table look-up, translate coded information into clear.
6. Produce a report according to a given hierarchy, with headings and totals where required.

Much of this is, of course, already provided by manufacturers' routines. All manufacturers provide (do they not?) sort routines, routines providing input and output between central processor and files, report program generators, tabulator simulators and the like. But a comprehensive program-generating-program is indicated which will extract, combine and capture all data which has passed or is passing through the computer. If a piece of data is used, even in a combined or implicit form, for one computer routine, it should by such a program be capable of extraction in the form required. The existence of such a program would be an indication that we sincerely hold to the first tenet of our philosophy. Such a program we do have. Rather shamefacedly I have to admit that we are guilty of yet another acronymic in calling this CRESTS, or Courtaulds Rapid Extract, Sort and Tabulate System. A crude and experimental version of CRESTS was written for the Honeywell 400 machine. With the decision to rewrite for the H2200 (a member of one of Mr. d'Agapayeff's Happy Families) came the loan of some Honeywell programmers to help us in our task. The CRESTS job was given to Tim Craig as the next programmer in line, with what interesting results can be judged from the following paper.