THE ROLE OF THE BEHAVIOURAL SCIENCES IN PROGRAMMING

B R Gaines

Monotype Holdings Ltd
Redhill
Surrey UK

B R GAINES is Technical Director of Monotype and Chairman of the Department of Electrical Engineering Science at the University of Essex. He is responsible for all technical activities and products throughout the Monotype Group; his main area of research is on human factors in computing, particularly on effective dialogue programming. Previously he was Technical Director of a timesharing service company and of a minicomputer manufacturing company, and consultant on computer systems engineering to a number of British and American companies. He has designed general purpose and special purpose computer hardware, and developed operating systems, languages and a wide variety of application packages.

THE ROLE OF THE BEHAVIOURAL SCIENCES IN PROGRAMMING

INTRODUCTION

The purchasers of computer systems are people, the specifiers of computer systems are people, the users of computer systems are people, the programmers of computer systems are people. If we have not always been acutely aware that the behavioural sciences are highly relevant to all aspects of computer systems implementation, then it is only because in attempting to create a computer 'science' we have modelled it on the stereotype of the physical sciences and quite deliberately forgotten the human aspects of what is still a frontier of human activity.

This paper is about the *role* of the behavioural sciences in computer system implementation; about the contributions which one might reasonably expect from psychology and sociology rather than the actual contributions they have made. Indeed, from the outset, it should be made clear that the potential contribution by far exceeds any so far made and that the lessons to be learnt are not necessarily direct solutions to problems, but, perhaps more importantly, meta-systemic aspects of methodology and experimentation; that there are techniques of experimental investigation that may throw light on particular problems, but there are also approaches that most certainly will not. Both psychology and sociology are more rich in case histories of the experimenter misled, or the theorist misinterpreting experiments, than they are in generally useful 'laws'. We are on very dangerous ground when we attempt to draw general inferences about people's behaviour from observations of specific cases, even under closely controlled experimental conditions (perhaps, *particularly* under such conditions man is *the* most highly adaptive animal and changes his behavioural colouration to his local circumstances more rapidly than a chameleon changes its optical colouration to its local base!).

As one of those who has strongly encouraged the growth of mutual awareness and interaction between behavioural and computer scientists, particularly in the last ten years through the editorial policy of the *International Journal of Man-Machine Studies*, I am acutely aware of the dangers as well as the potential rewards of introducing techniques and concepts from the behavioural sciences into computer science and its practical application. It is easy to put into effect a 'human experiment' in any new venture: we are installing an interactive timesharing partition into what was previously a batch processing environment. What do the users expect to gain? What actually occurs in terms of programmer productivity etc? We find that programmers expect much but gain nothing and hence we conclude that interactive access has been over-sold and is actually of little benefit; we do not ask whether it is because the interactive partition is up only 2 hours a day, provides no continuity of on-line storage and has a response time of 15 seconds!

The majority of real-world situations provide little opportunity for experimental studies that could lead to significant conclusions. In general, also, moving to the 'laboratory' situation and creating a micro world that simulates in some aspects the real-world again is of limited benefit in leading to significant results applicable outside the laboratory. This is not to say that both approaches have not been used meaningfully in important and useful studies, but, invariably, such studies also have a major component of penetrative intuition grounded on practical experience in their design and interpretation. The methodologies of the behavioural sciences provide no substitute for thought, doubt, introspection and informal observation. These are the substance of our understanding of human behaviour and the formality of empirical experiment follows it rather than precedes it. Perhaps this is obvious, particularly to an audience of practical systems designers, but there is a danger that the behavioural sciences, in presenting themselves as modelled on the physical sciences, may appear to offer a route to understanding that does not exist. These sciences themselves are unsure of their foundations and this is significant to their application. It by no means undermines their relevance since it is the same problem of the essential complexity of real-world systems that we face in both the behavioural and computer sciences.

In the next section I shall develop a framework for understanding the role of the behavioural sciences in computer sciences, and finally give some illustrations of the types of study to date.

THE VIRTUAL MACHINE HIERARCHY AS A FRAMEWORK FOR HUMAN FACTORS

In charting the quicksands of current human studies in computer science one needs some kind of framework in which to place the results. Human factors appear in virtually all aspects of computing; some machines are 'easier to program' than others; some languages are 'easier to learn' than others; some interactive programs seem 'more natural' than others. Are these different phenomena and must one treat human problems in computing piecemeal? Or, is there some underlying model which encompasses a range of phenomena?

At one extreme one may study the human aspects of machine architecture, both of special purpose and general purpose computers, asking such questions as what makes a computer an 'analogue computer' (006), and what makes one instruction set more comprehensible than another? (007). At the other extreme one may study the problem of programming dialogue between naive users and the computer and look for general rules relating to effective dialogue programming (009,010). The results suggest that there is a strong commonality between the principles that are drawn from hardware, software, and end-user dialogue situations, that gives hope that the same underlying psychological phenomena are relevant to these diverse aspects of computing.

The concept in computing science that has been most fruitful in enabling an integrated and uniform view to be taken of hardware and software is that of a *virtual machine*. It is a term which originally arose in the context of emulation, when it was necessary to speak of one hardware processor being emulated by a program on another. Functionally the combination of hardware and software in one machine is equivalent to the hardware alone of the other; it is a 'virtual machine' equivalent to a real one. It was natural for the term to be quickly extended to situations where no hardware 'machine' existed or was even likely to exist: operating systems add functionality to a machine and the calls on an operating system may be regarded not as links to other software but as the use of a machine with an enhanced instruction set; similarly, the run-time system of a FORTRAN compiler may consists of a set of library routines that are called by the main program. These may alternatively be regarded as instruction set enhancements generating a new virtual machine, a 'FORTRAN machine'.

Research on virtual machines as such (014) has tended to concentrate on effective emulation, nowadays not just of hardware but of the hardware plus operating system. However, it is very useful to extend the use of the term to its logical conclusion and think of any system seen from the outside as a 'black box' defined not by the way it is implemented but solely by its actual functionality. I have often quoted John Cleary's remark at EUROCOMP '74 that: 'The only virtual machine of interest to a user ... [is] one that has abolished all lower levels of system and presents an *understandable* and *sympathetic* face to its user' and have developed formal rules for the virtual machine that a naive user needs to see for effective man-computer communication (009) — even concepts such as 'sympathy' and 'understanding' have well-defined explanations. Between the hardware level and the end-user applications level there is a complete and, nowadays often very lengthy, hierarchy of virtual machines. The dialogue you have at a terminal in order to learn statistics is written as a program in an 'author language' that is itself written in a 'string-processing' language that is itself written in 'FORTRAN' that itself compiles to an 'intermediate assembly language' that runs under an 'operating system' in the 'instruction set' of a HAN269 computer that is emulated in the 'microinstruction' of.... One wonders sometimes whether there is anything out there actually doing something, or whether it is not an infinite regression of virtual machines, particularly when the response time exceeds 30 seconds!

Figure 1 shows such a hierarchy of virtual machines. Note that it is not just a linear chain of levels but rather a true branching structure in that the same virtual machine may run many different programs, and that the hierarchy is embedded in a lattice in that different virtual machines may run the same program. Note also, however, that there is nothing in Figure 1 to indicate that virtual machines at different levels have any common properties, that there is some degree of recursion in the hierarchy. This appears only when we examine the human factors relating to the virtual machine.

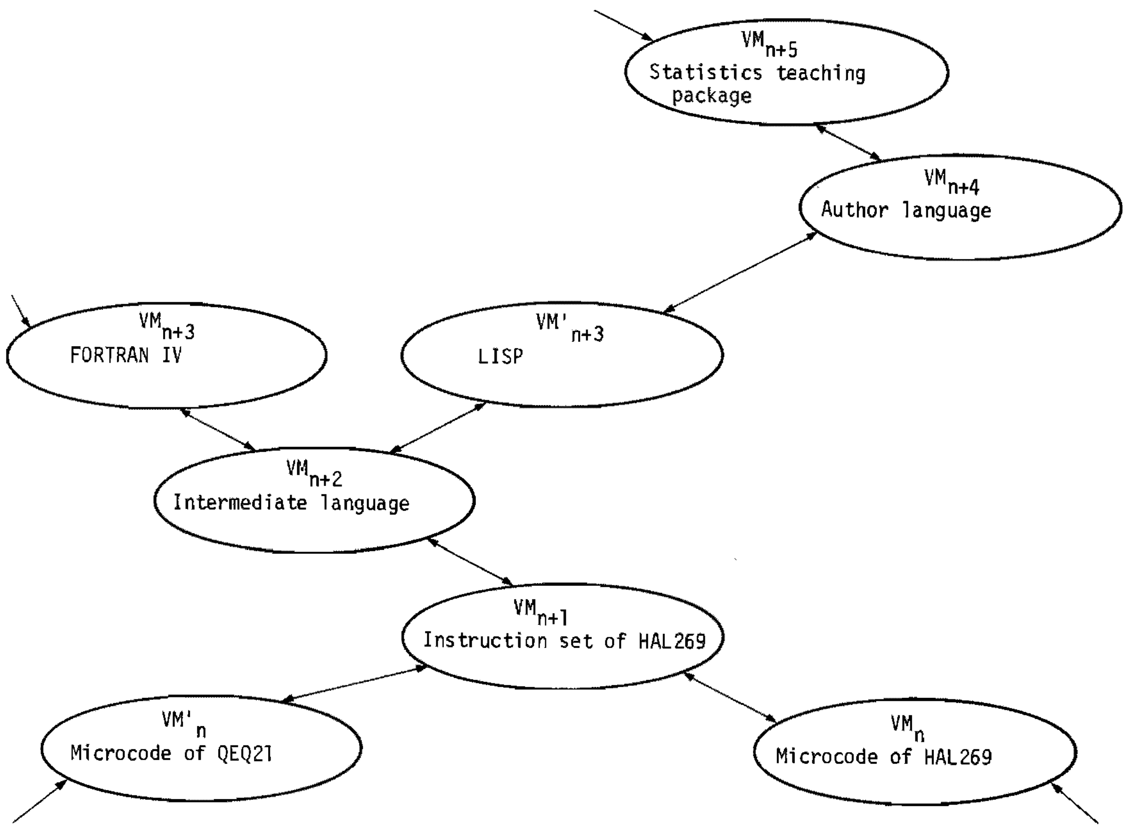

Figure 1: A virtual machine hierarchy

The interrelationship between machine, problem and programmer

Consider now the relationship between two levels of virtual machine in Figure 1 in terms of implementation. At level *n*+4, for example, the 'author language' has to be implemented in LISP, i e, the facilities provided by one level of virtual machine are used to implement the next higher level. We may clearly ask technical questions about whether LISP is a suitable language in which to write the required author language, does it provide suitable data types, structures, and operations upon them? However, if we try to relate the discussion in the literature on 'structured programming' to such a technical relationship between virtual machines we rapidly find that the two-part relationship expressed is inadequate to capture many significant concepts. In discussing program*ming* we have to bring in the programm*er* and we find that he (she or it) has also a relationship to each of the virtual machines — is he fluent in LISP? how does he think of the author language? We have actually to consider the *three-part* relationship shown in Figure 2 between:

- The programmer
- The virtual machine which provides implementation facilities
- The virtual machine which is the problem whose solution is to be implemented.

We can now see the programmer as a medium through which a virtual machine at one level is transformed to one at another (less mystically, a virtual machine at level *m* may be regarded as one at level *m*-1 plus a program!).

If we now bring in the three-part relationship of Figure 2 into the virtual machine hierarchy of Figure 1, we get the diagram of Figure 3. Two features of the hierarchy are now immediately apparent.

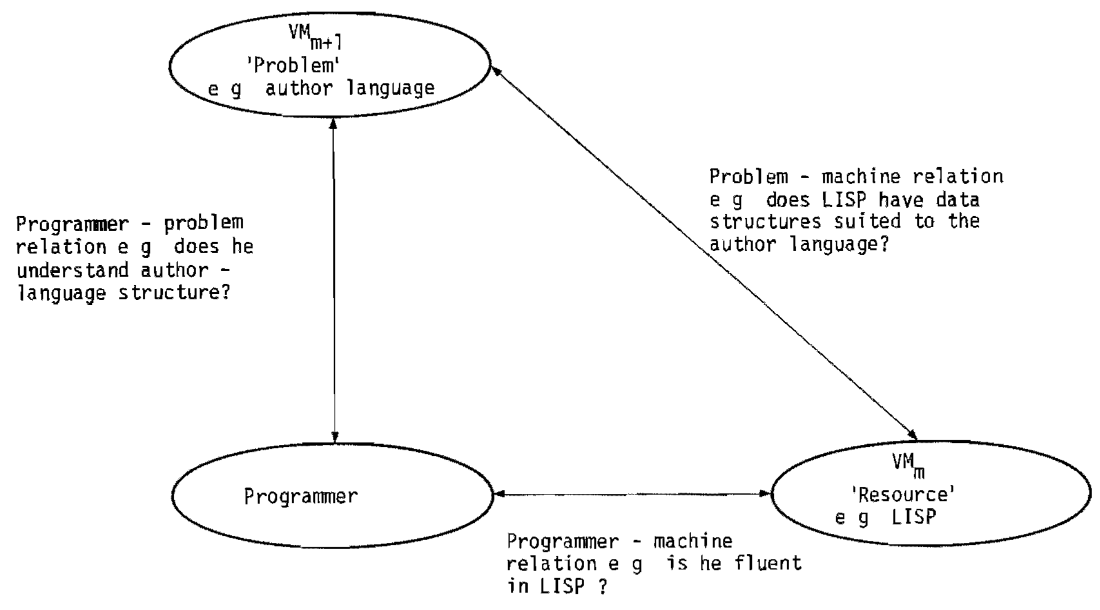
VM_{m+1}
'Problem'
e g author language
Problem - machine relation e g does LISP have data structures suited to the author language?
Programmer - problem relation e g does he understand author - language structure?
Programmer
VM_m
'Resource'
e g LISP
Programmer - machine relation e g is he fluent in LISP ?

Figure 2: The three-part relationship between programmer, problem and virtual machine

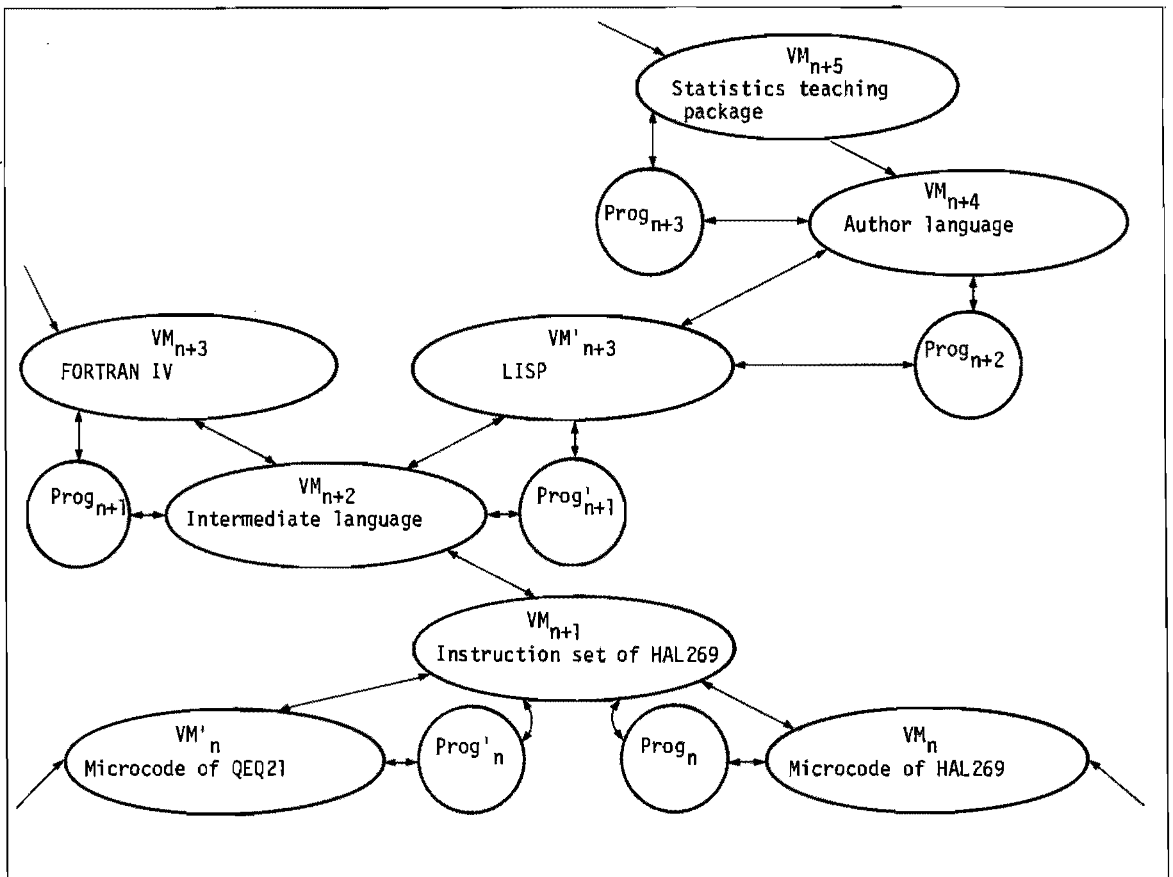

Figure 3: Human factors in a virtual machine hierarchy

First, the *isolation* between levels that is possible: the programmer at level *m* has to consider only the machines at level *m* and *m*+1: one given to him as a resource and the other as a problem to be solved using the resource. He does *not* have to consider virtual machines at lower or higher levels, i e, how the resource he has been given has been implemented, or how the resource he is creating will be used. Or, if he does have to consider such factors then the degree of isolation superficially apparent has not really been achieved; the problem is inadequately specified; the resource given is inadequate in definition or quality.

Secondly, the *recursive nature* of the virtual machine hierarchy becomes apparent: at each level there is a 'programmer' who is an intermediary between resource and problem — he is an algorithmic problem-solver and, whilst there are differences between the problems and resources at different levels, there are also very strong similarities. A good example of this is the command structure at each level whereby the resources are controlled. Consider virtual machine A with 400 commands each of which is very significant and useful but has to be understood in isolation. Contrast this with virtual machine B which has also 400 commands that may, however, be regarded as 8 main functions each of which has 5 modes of operation on 10 different types of data structure. The two machines A and B present very different problems of learning and comprehension to the programmer — it is the difference between remembering 400 things and 8+5+10=23 things. *This principle applies at every level of the virtual machine: to the micro-instruction set, to the main instruction set, right down to the information entry and access commands used by the ultimate end-user, e g, a nurse in a hospital running a patient administration system.*

Similar universality and transfer of experience, and principles derived from it, apply to a range of phenomena relating to human factors in the problem-programmer-machine triad at all levels. The virtual machine framework of Figure 3, also encompassing this three-part relationship as it occurs at each level, seems to provide an adequate conceptual base for the discussion of nearly all aspects of human factors in 'structured programming'. Clearly it has inherent within it models of both 'top-down' and 'bottom-up' approaches to overall system design, and it also makes ostensive notions of 'modularity' and its advantages.

Dijkstra (003) notes the fundamental significance of the virtual machine in his original paper on structured programming and it has become a significant theme in later work (018). As Cremers and Hibbard (002) recently remarked: 'Systematic software design can be based on the development of a hierarchy of virtual machines, each representing a level of abstraction of the design process'. The virtual machine hierarchy is an intuitively meaningful concept which can also be modelled formally (003), and it has also been suggested that the role of the programmer in it can also be encompassed through a suitable formalism (008).

Explicit inclusion of the programmer within the framework also allows for, and models, the conflicts that can arise between what is clearly good practice and what actually occurs in practice: the fluid dynamicist who writes a natural language dialogue program in FORTRAN, despite the very poor problem-machine relationship, because his knowledge of FORTRAN is very good and his knowledge of more suitable languages is virtually non-existent; or, conversely, the programmer who creates a highly inefficient and ineffective implementation when the problem-machine relationship is actually very well matched because he himself has a very poor understanding of the intrinsic requirements of the problem; and so on.

DIRECTIONS IN HUMAN FACTORS RESEARCH

I have dwelt at length on a conceptual framework for human factors in programming because to my mind this is the most urgent prerequisite for advances in the next decade. We do now have a body of published material on such human factors (001,004,005,009, 010,011,012,013,015,016,017,019-031) and, both as a research area and as a topic of direct commercial significance, studies in this area are coming very much into vogue. There is very much more to be gained from these studies as a whole if they can be related to one another and compared with one another even though their practical contexts and applications are very different. It is easier and safer to interpolate in the gaps of a diagram such as Figure 3 than to extrapolate from one isolated study to a different context.

To give a feel for the type of work that has so far been undertaken I will give some examples from the literature which illustrate different approaches.

Sime and his group at Sheffield University (026,027,028) have taken a classical experimental psychology approach to the study of human aspects of programming language constructs and performed laboratory experiments in which groups of subjects are compared in performance and nature of errors in programming up the same problem using different families of allowed constructs. Their methodology is of interest whenever specific points of major significance have been isolated, e g, the merits of the 'GOTO' construct compared with alternatives. Laboratory experiments are, of course, notoriously difficult to extend to the real world. However, the very exercise of constructing a laboratory situation, even as a gedanken experiment, is invaluable in clarifying one's thinking and arguments. What are the key features? In what context must they be tested? What interfering factors must be eliminated? Is there interaction with the knowledge and experience of the programmer? and so on. Many a loose and woolly argument has either become transparently ridiculous or simplified to the point of tautology when exposed to the discipline of experimental design. The operational viewpoint – how can I actualise precisely what I am talking about and nothing else – is a necessary technique to effective human factors thinking.

Most other studies reported in the literature are *ethological* and/or *introspectionist*. In the ethological approach one studies actual animals going about their natural habitat, in this case programmers programming, users using, and so on. For interactive users it is possible to take transcripts of the interaction automatically which enable the programmers' behaviour to be analysed in great detail. Even in a batch environment it is possible to follow the successive stages of programming debugging and even to analyse the comparative differences between successive generations of a program automatically (015). At a coarser level the logged statistics of use of a computer may be used to provide valuable information on the pattern of activities of users as a whole and the differential behaviours of different classes of user (004,016). This often generates very interesting results (e g, Doherty (004) reports that for each additional second of system response time the user response time also increases by one second) which are by no means easy to interpret in terms of individual action (for example, does the user's attention wander individually as a random walk, or do some users get fed up and do some intervening task – either phenomenon might account for the data) and this often points to the type of more detailed laboratory experiment that would be worthwhile (e g, monitor individual users in a regime of controlled variation of system responsiveness).

The introspectionist approach through self-analysis of one's own experience, behaviour and attitudes, played a key role in the early development of psychology but fell into disrepute as the behaviourist view came to hold sway. Introspection is invaluable in practice, however, as an adjunct to the observation of others – we gain insight into the behaviour of others by mentally putting ourselves in their place and relating their behaviour to our own experiences. The literature on high level skills such as chess has a major component in the self-analysis of chess masters, and the literature on programming skills has a similar component based on the self-analysis of experienced programmers. Again such introspection often suggests more objective ethological observations of others or laboratory experiments. There are also techniques for the formal gathering and analysis of introspective data through questionnaires, and these have been used in important studies of human factors in computing (005).

CONCLUSIONS

Programming is a human skill and the behavioural sciences have a part to play in our understanding the process of programming and improvements in programming facilities and techniques. The commercial significance of this has been appreciated and, for example, various IBM research laboratories figure largely in any list of published research on human factors in programming. The behavioural sciences themselves offer no royal road to understanding but can contribute basic features of human behaviour, particularly cognitive skills, and a wealth of material on the methodology of experiments on, and observation of, human behaviour. The fundamental framework of a virtual machine hierarchy which has proved a dynamic concept in structured programming may be extended to take account of human factors and this throws new light on important features of the concept. Future developments in the formal analysis of this framework, coupled with human experiment and observation in the light of experience, have a vital role to play in our coming to grips with the complexities of modern computer systems implementation.

REFERENCES

001 BROOKS R
Toward a theory of the cognitive processes in computer programming
Intl J Man-Machine Studies vol 9 pp 737-751 (Nov 1977)

002 CREMERS A B and HIBBARD T N
Formal modelling of virtual machines
IEEE Trans Software Eng vol SE-4 pp 426-436 (Sept 1978)

003 DIJKSTRA E W
Notes on structured programs
In *Structured Programming*
Academic Press New York (1972)

004 DOHERTY W J
Human factors: impact on interactive programming
SHARE Meeting Cambridge UK (Sept 1977)

005 DZIDA W, HERDA S and ITZFELDT W D
User-perceived quality of interactive systems
IEEE Trans Software Eng vol SE-4 pp 270-276 (July 1978)

006 GAINES B R
Varieties of computer — their applications and interrelationships
IFAC Symp Budapest (April 1968)

007 GAINES B R
Computer technology and its utilization today and tomorrow
Proc Conf on *Small computer applications in industry* Nat Eng Lab East Kilbride UK (March 1973)

008 GAINES B R
Analogy categories, virtual machines and structured programming
Lecture Notes in Comp Sci vol 34 pp 691-699 GI-5 Jahrestagung (1975)

009 GAINES B R
Programming interactive dialogue
Pragmatic Programming & Sensible Software pp 305-320 On-line Uxbridge (1978)

010 GAINES B R and FACEY P V
Some experience in interactive system development and application
Proc IEEE vol 63 pp 155-169 (June 1975)

011 GANNON J D
An experiment in the evaluation of language features
Intl J Man-Machine Studies vol 8 pp 61-73 (Jan 1976)

012 GILB T
Software metrics
Studenlitteratur Sweden (1976)

013 GILB T and WEINBERG G M
Humanized input
Winthrop Publishers Cambridge Mass
(1977)

014 GOLDBERG R P
Survey of virtual machine re-search
Computer vol 7 pp 34-35
(June 1974)

015 GOULD J D
Some psychological evidence on how people debug computer programs
Intl J Man-Machine Studies vol 7
pp 151-182 (March 1975)

016 HARALAMBOPULOS G and NAGY G
Profile of a university computer user community
Intl J Man-Machine Studies vol 9
pp 287-313 (May 1977)

017 HOC J M
Rôle of mental representation in learning a programming language
Intl J Man-Machine Studies vol 9
pp 87-105 (Jan 1977)

018 HORNING J J and RANDELL B
Process structuring
Comp Surv vol 5 pp 5-30
(1973)

019 MARTIN J
Design of man-computer dialogues
Prentice-Hall New Jersey (1973)

020 MILLER L A
Programming by non-programmers
Intl J Man-Machine Studies vol 6
pp 237-260 (March 1974)

021 MILLER L A and THOMAS J C
Behavioural issues in the use of inter-active systems
Intl J Man-Machine Studies vol 9
pp 509-536 (Sept 1977)

022 NAGY G and PENNEBAKER M C
A step towards automatic analysis of student programming errors in a batch environment
Intl J Man-Machine Studies vol 6
pp 553-578 (Sept 1974)

023 REISNER P
Use of psychological experimentation as an aid to the development of a query language
IEEE Trans Software Eng SE-3 pp 218-229
(May 1977)

024 SHNEIDERMAN B
Exploratory experiments in programmer behaviour
Intl J Comp Infor Sci vol 5 pp 123-143
(June 1976)

025 SHNEIDERMAN B
Measuring computer program quality and comprehension
Intl J Man-Machine Studies vol 9
pp 465-478 (July 1977)

026 SIME M E, GREEN T R G and GUEST D J
Psychological evaluation of two condit-ional instructions used in computer languages
Intl J Man-Machine Studies vol 5
pp 105-113 (Jan 1973)

027 SIME M E, GREEN T R G and GUEST D J
Scope marking in computer conditionals — a psychological evaluation
Intl J Man-Machine Studies vol 9 pp 107-118 (Jan 1977)

028 SIME M E, ARBLASTER A T and GREEN T R G
Reducing programming errors in nested conditions by prescribing a writing procedure
Intl J Man-Machine Studies vol 9 pp 119-126 (Jan 1977)

029 THOMAS J C and GOULD J D
A psychological study of query by example
Proc '75 Comp Conf pp 439-445 (May 1975)

030 WEINBERG G M
The psychology of computer programming
Van Nostrand Reinhold (1971)

031 YOUNGS E A
Human errors in programming
Intl J Man-Machine Studies vol 6 pp 361-376 (May 1974)