Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

23

Programming Methodology

Lecture Notes in Computer Science

Edited by G. Goos and J. Hartmanis

23

Programming Methodology

4th Informatik Symposium, IBM Germany
Wildbad, September 25–27, 1974

Edited by Clemens E. Hackl

Springer-Verlag
Berlin · Heidelberg · New York 1975

Editorial Board: P. Brinch Hansen · D. Gries
C. Moler · G. Seegmüller · N. Wirth

Prof. Dr. Clemens E. Hackl
IBM DEUTSCHLAND
UV Wissenschaft
7 Stuttgart 80
Pascalstr. 10^
BRD

Library of Congress Cataloging in Publication Data

Informatik Symposium, 4th, Wildbad im Schwarzwald, Ger., 1974.
Programming methodology.

(Lecture notes in computer science ; 23)
"Sponsored by IBM Germany and the IBM World Trade Corporation."
Bibliography: p.
Includes index.
1. Programming (Electronic computers)--Congresses. I. Hackl, Clemens E., ed. II. IBM Deutschland. III. IBM World Trade Corporation. IV. Title. V. Series.
QA76.6.I47 1974 001.6'42 74-34362

AMS Subject Classifications (1970): 00A10, 02G05, 02G10, 68A05, 68A10, 68A20, 68A30, 68A40
CR Subject Classifications (1974): 4.0, 4.12, 4.20, 4.22, 4.30, 4.6, 5.20, 5.23

ISBN 3-540-07131-8 Springer-Verlag Berlin · Heidelberg · New York
ISBN 0-387-07131-8 Springer-Verlag New York · Heidelberg · Berlin

This work is subject to copyright. All rights are reserved, whether the whole or part of the material is concerned, specifically those of translation, reprinting, re-use of illustrations, broadcasting, reproduction by photocopying machine or similar means, and storage in data banks.

Under § 54 of the German Copyright Law where copies are made for other than private use, a fee is payable to the publisher, the amount of the fee to be determined by agreement with the publisher.

© by Springer-Verlag Berlin · Heidelberg 1975. Printed in Germany.

Offsetdruck: Julius Beltz, Hemsbach/Bergstr.

PREFACE

The papers in these proceedings were presented at the 4th Informatik-Symposium which was held in Wildbad, Federal Republic of Germany, from September 25 - 27, 1974. The symposium was organized by the Scientific Relations Department of IBM Germany and sponsored by IBM Germany and the IBM World Trade Corporation.

The aim of the Informatik-Symposia is to strengthen and improve communication between universities and industry by covering a subject in the field of computer science as well from a university as from an industrial point of view.

Following last year's subject of computer structures the emphasis in this conference was placed on programming methodology which has become a field of increasing activity during the last years. Like in hardly any other segment in computer science problems are related to research, development, education and application with progress depending both on advances in theory and on increased practical experience.

By organizing this symposium it was tried to cover this broad spectrum of programming methodology. At the beginning problems and experiences of production programming in an industrial environment were presented. Aspects for the development of large systems, organizing for structured programming, investigations about the reliability of programming systems and error analysis and error causes in production programming were covered.

After presenting the industrial aspects system programming was considered from an university point of view. Problems in education, in language design for systems programming and general aspects of software engineering were addressed.

In the following lectures emphasis changed from product development to subjects in advanced development and research. New approaches for program testing, new concepts about reasoning in program synthesis and methods of interprocedural analysis were presented.

A subject of particular importance in advanced programming seems to be functional or nonprocedural programming. The strictly sequential character of a program is relegated to the background in favour of a precise description and formulation of the problem to be solved. A change from a procedure oriented programming to a description oriented programming could be a final goal.

In concluding the symposium contributions were presented which were related to the formal definition and representation of programs, the description of mathematical structures in programming languages and to the axiomatic foundation of programming languages.

Finally, we would like to thank all the lecturers for their contributions and for the very valuable advice and assistance given during the preparation of the symposium.

Stuttgart, October 24, 1974

Gerhard Hübner
Manager Scientific Relations
IBM Germany

Clemens E. Hackl
Symposium Chairman

此为试读,需要完整PDF请访问: www.ertongbook.com

CONTENTS

ON THE DEVELOPMENT OF SYSTEMS OF MEN AND MACHINES

H. D. Mills, IBM Corporation and The Johns Hopkins University

With Appendix by R. C. Linger, IBM Corporation

ABSTRACT

We formulate the development of systems of men and machines as a programming problem for multiprocessing in which some processors are men and some are machines. In this way, users guides, training courses, etc., are determined from processing requirements, just as machine specifications, which are consistent with the objectives of the total operation. An Appendix illustrates this idea in miniature for a supermarket checkout operation.

Systems of Men and Machines

Our topic is the architecture, implementation, and operation of large systems of men and machines in some definite and coherent enterprise -- managing a business, operating an airline reservation system, running a government agency, getting men to the moon and back, etc. In such systems there are many kinds of men (and women) -- managers, clerks, specialists of various kinds, and machine tenders; there are also many kinds of machines -- computers, terminals, sensors, actuators, communications equipment, etc.

Ordinarily computer programming is regarded as part of the machine side of the system, and programmers as part of the machine tenders. We bring a different view -- that the architecture of the operations of the enterprise is programming, as well. Our thesis is that modern principles of programming -- forced on us out of necessity in dealing with machines of much logical capability but no common sense -- can play as vital a role in bringing systematic discipline and standards into all phases of large systems development.

For this point of view we escalate the concept of programming to that of providing comprehensive instructions for either man or machine activities. The operations of an enterprise becomes a multiprocesssing operation of men

and machines; then the architecture of the operations defines the configurations and types of men and machines in the system, and the programs which direct each type of man and machine. For example, a users guide becomes one part of a cooperating system of programs operating in separate processors (the user and a machine).

Of course, the characteristics of man and machine are quite different in such a system, just as machines are different among themselves. And yet their architectural properties can be treated in a uniform way. For example, in deciding on a particular machine requirement, various considerations of physical or logical capability will arise, as well as whether the requirements can be met with off-the-shelf equipment; these same considerations apply to a man requirement, in common terms -- e.g., how many letters can a postal clerk sort in an hour, and can this be done with minimal training, etc.

It is well understood that men and machines do well at quite different things. Machines are good at doing what they are told to do, very rapidly and accurately. Men are good at using common sense -- even disobeying instructions when they are obviously misconceived; at pattern recognition -- discovering information by no special or dependable process; and at invention -- creating a new idea for the enterprise to act on. One simple, but very important form of pattern recognition is the translation of human speech to and from machine readable text. This is routine for clerical activities that interface with persons outside the enterprise -- e.g., in an airline reservation system.

It may be asked how the concept of programming applies to men, which operate so differently and unpredictably, compared to machines. It applies by noting that programs are used in a local way by machines -- i.e., at this moment, under these conditions, do this next. A good deal of programming on the man side is already subsumed under general instructions and common sense -- if the telephone rings, answer it; if you want to execute a program, keypunch a job deck and submit it to the operator. We have no intention of explicitly programming human activities now done by general instructions and common sense; the typical level of human programming envisioned is that which is found in users guides, operating instructions, etc., associated with machine operations.

Principles of Programming for Men and Machines

The recent, twenty-five year, history of computer programming has seen an explosive and traumatic growth of human activity and frustration in attempting to realize the promise and potential of electronic and electromechanical devices for processing data in science, industry and government. Out of this history has come the stored program computer; programming languages, compilers, and libraries; and new technical foundations for programming computers, e.g., as propounded by McCarthy [4], Dijkstra [1], Hoare [3], and Wirth [10]. In this period the computer has been the center of attention. In the beginning, the numerical computing power was so great, compared to manual methods, and the availability so limited, that men readily adapted to this new tool -- from decimal to binary, from equations to algorithms. But in a short time, the remarkable possibilities for more general data processing (nonnumerical) were realized, and a new industry was born in just a few years. In the later part of this period (up to now) the large data processing systems appeared -- management information systems, airline reservation systems, space tracking systems, etc. Even then, although human factors were considered, these systems were conceived primarily as data processing systems, which responded to users. But in our proposed perspective, the users are as much a part of the multiprocessing enterprise as the machines and their programs.

Thus, whereas the computer has forced us to find more effective programming principles than we would otherwise have, it has also warped our sense of perspective. By this time, simple human factors questions are in better focus. There is enough data processing power available to invest part of it in creating more human-like interfaces than binary and machine code. But in extending programming to instructing men, with their entirely different characteristics, additional ideas and principles are needed, such as

1. Languages. The programming languages used in the architecture of systems of men and machines need be near natural languages. The difficulties of processing natural languages are well known and that is not proposed. Rather, what is proposed is a "naturalization" of processable programming languages which is close enough for use as a dialect of natural language by nonprogrammers of the enterprise. Sammet [8] and Zemanek [11] discuss both sides of this.

2. Procedures. The concept of procedure should be extended to include indefinite procedures where it is not possible or desirable to define them. In simple terms, "find values for these variables so that these equations hold" (Wilkes discusses this in [9]), or more complex, "make sure no one's feelings are hurt".

3. Interactions. The principal subject of the architecture is multiprocessing -- the conduct of the operation of the enterprise through programs distributed to men and machines. The creation of such programs in an orderly, systematic way will require a new and fundamental development beyond programming principles for synchronous operations. The idea of the Petri net [7] may be an embryonic step in such a development. Dennis illustrates Petri nets as multiprocessing control mechanisms, in [2].

Architecture Principles

A system depends on its components. The architecture of a system specifies the types of men and machines required, as well as how they are to interact as a system operation, i.e., the selection and arrangement of the system components, as well as detailed instructions for their behavior. In the case of machines, the usual considerations apply -- define feasible requirements, either already embodied in existing machines, or possible with special development where justified; tradeoffs and comparisons with alternative approaches, even with manual approaches where feasible, etc. In the case of men, the system architect is frequently shortsighted. There is a reason; it is much more difficult to predict a human performance than a machine performance. Sometimes the human fails to live up to a requirement. But often the human will exceed a requirement in a totally unexpected way, by acquiring a skill not imagined possible beforehand. This is happening with computer programmers right today, who are beginning to program with a precision not believed possible five years ago. It happened with typing when touch typing was introduced early in this century.

In retrospect, it is easy to identify a pitfall in overestimating machine possibilities, leading to a common scenario in many large systems in recent history, which put a "man in the loop" at the last moment, with marginal operational results. In such a case, the operation was originally planned as completely automatic, depending on some key algorithm (often involving some form of pattern recognition); as a result of the planning, the data processing functions surrounding the algorithm were developed in parallel

此为试读,需要完整PDF请访问: www.ertongbook.com

according to a general system design; then at the last moment, the performance of the key algorithm proves inadequate, and the man is brought into the loop, with two costs:

1. The human factors are bad (e.g., interacting with programs which require long, fixed argument lists); these can be fixed up.

2. There is a lost opportunity in not having a better trained man in the loop. The effort on algorithm development is frequently different than that required for insight development for a human executing an indefinite procedure, and the time is lost, anyway.

These operational experiences lead to the following principles of systems architecture:

1. Components. Regard men and machines as equal status components for system operations with equal requirements for development, state-of-art projections, and improvement, according to their own characteristics.

2. Evolution. Plan on the unexpected, by well-structured interfaces, that permit the replacement of components by improved versions which perform identical functions more effectively. Parnas [6] deals with this interchangability by axiomatizing such interfaces.

3. Integrity. Value system integrity above all else, by requiring that the multiprocessing operation of men and machines be described and scrutinized according to the best principles of programming -- particularly with respect to methods of specification and validation of programs. Note especially the technique defined by Wirth [10] and Mills [5].

In an Appendix dealing with a minature problem, R. C. Linger illustrates the idea of programming an operation through a set of abstract processes which only later are specialized to either men or machines, depending on their requirements. It is an easy transition from man as a user to man as a processor and yet it seems a critical one in providing system coherence and integrity with respect to a given operation.

Literatur

[1] O. -J. Dahl, E. W. Dijkstra and C. A. R. Hoare, Structured Pr
Academic Press, London. 1972.

[2] J. B. Dennis, "Concurrency in Software Systems", Lecture Not
in Economics and Mathematical Systems, 81 Advanced Course on
Software Engineering (Ed. F. L. Bauer), Springer-Verlag, Ber
Heidelberg, New York. 1973.

[3] C. A. R. Hoare, "An axiomatic basis for computer programming
CACM 12. 1969. pp. 576-580, 583.

[4] J. McCarthy, "A basis for a mathematical theory of computati
Computer Programming and Formal Systems, (Eds. P. Braffort a
D. Hirschberg), North-Holland Publishing Company. 1963. pp

[5] H. D. Mills, "The new math of computer programming", CACM 1
1975. (To appear).

[6] D. L. Parnas, "A technique for software module specificati
examples", CACM 15, 5. May 1972. pp. 330-336

[7] C. A. Petri, Communications with Automata. Supplement 1 t
Technical Report RADC-TR-65-377, Vol. 1, Griffiss Air For
Base, New York. 1966. (Originally published in German:
Kommunikation mit Automaten, University of Bonn, 1962).

[8] J. E. Sammet, "The use of English as a programming langua
CACM 9, 3. March 1966. pp. 228-230.

[9] M. Wilkes, "Constraint-type statements in programming la
CACM 7. 1964. pp. 587-588.

[10] N. Wirth, Systematic Pro
Hall, Englewood Cliffs, New Jersey. 1973.

[11] H. Zemanek, "Semiotics and programming languages", CACM
3. March 1966. pp. 139-143.

x

ARKET CHECKOUT AS A MAN-MACHINE MULTIPROCESSING ACTIVITY

nger, IBM Corporation

the problem of specifying a design for a super market checkout
n. Several possibilities come to mind; a man with an adding machine
box, a man with a cash register, a man with an OCR device which reads
prices, etc. Whatever the final configuration, we can begin our
lth a procedural description of the checkout process itself. In this
e dynamics of the process are of central interest. That is, we do
h with a static description of components which must somehow fit
in a presumed system, but rather begin with a process which can be
work, and derive required (and possibly alternate) component config-
of men and machines from it. Our tentative first refinement is:

```
start checkout
   open checkout station at 9 a.m.
   do while before 5 p.m.
      checkout next customer, if any
   od
   close checkout station
stop
```

actions of men and machine are abstracted; we cannot determine
what, but can agree that the description seems reasonable so far.
refinement might be:

```
start checkout
   establish man on duty at 9 a.m.
   power up machines at 9 a.m.
   do while before 5 p.m
      ac[illegible] next [illegible] any
      total cost of all items
      inform customer of total
      accept payment
      present change, if any, to customer
      bag all items
   od
   balance cash total with sales total
   power down machines
stop
```

Here the functions of man and machine begin to emerge; presumably they will cooperate to "total cost of all items" through manual keystroke entry or OCR input, and the man will "bag all items," without mechanical assistance. We identify "before 5 p.m." as a man predicate, in view of the human propensity for clockwatching, which hopefully yields to common sense to complete checkout of customers waiting in line at quitting time! The possibility of "exception processing" arises in this procedure, as when the machines break down, or a coffee break is desired. These situations are analogous to the interrupt handling facilites of modern computers.

An aspect of the activity distribution between man and machine can be explored through elaboration of the process to "total cost of all items:"

```
start total cost of all items
   initialize subtotals to zero
   do while items remain
      if item type is produce then
         add price to produce subtotal
      else
         if item type is meat then
            add price to meat subtotal
         else
            add price to grocery subtotal
         fi
      fi
   od
   add subtotals to find total cost
stop
```

The cooperating give and take actions of man and machine appear as an *abstract* process here; "items remain" is likely a man predicate, while the "item type" expressions appear to be machine predicates set by man. The subtotal accumulations are best handled as machine functions.

We can establish concrete procedures for man and machine by defining an interface between them. In illustration, consider an electromechanical cash register with control keys as follows:

0, 1,...,9	price digits
.	decimal point
I	initialize (for new customer)
P	produce
M	meat
G	grocery
T	total

For this interface, the man procedure becomes

```
start total cost of all items -- man part
   push I
   do while items remain
      get next item
      push digit and decimal keys for price
      if item type is produce then
         push P
      else
         if item type is meat then
            push M
         else
            push G
         fi
      fi
   od
   push T
stop
```

and the corresponding machine procedure is:

```
start total cost of all items -- machine part (I key depressed)
   set produce subtotal to zero
   set meat subtotal to zero
   set grocery subtotal to zero
   do until T key
      wait for T|P|M|G key
      if ¬ T key then
         read and clear price register
         if P key then
            add price to produce subtotal
          else
            if M key then
               add price to meat subtotal
            else
               add price to grocery subtotal
            fi
         fi
      fi
   od
   add produce, meat, grocery subtotals
   display result
stop
```

A different set of procedures derive from an alternate interface, as with a cash register using OCR input, and equipped with only I and T keys. In this case the man procedure simplifies to

```
start total cost of all items -- man part
   push I
   do while items remain
      pass item label over OCR window
   od
   push T
stop
```

and the matching machine procedure must extract both item type and price from encoded labels.

We observe that the man procedures above can evolve naturally into users guides and training courses, possibly containing instructions no machine would understand, as, "Don't be distracted while pushing price keys." The machine procedures give a explicit basis for electromechanical component design, or executable procedures in the case of programmable devices.

此为试读,需要完整PDF请访问: www.ertongbook.com