QI QUAESTIONES INFORMATICAE

Vol. 3 No. 3 August, 1985

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

QUAESTIONES INFORMATICAE

An official publication of the Computer Society of South Africa and of the South African Institute of Computer Scientists

'n Amptelike tydskrif van die Rekenaarvereniging van Suid-Afrika en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Editor: Prof G. Wiechers
INFOPLAN
Private Bag 3002
Monument Park 0105

Editorial Advisory Board

PROFESSOR D. W. BARRON
Department of Mathematics
The University
Southhampton S09 5NH
England

PROFESSOR J. M. BISHOP
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg 2001

PROFESSOR K. MACGREGOR
Department of Computer Science
University of Cape Town
Private Bag
Rondebosch 7700

DR. H. MESSERSCHMIDT
IBM South Africa
P.O. Box 1419
Johannesburg 2000

MR. P. C. PIROW
Graduate School of Business
Administration
University of the Witwatersrand
P.O. Box 31170
Braamfontein 2017

MR. P. P. ROETS
NRIMS
CSIR
P.O. Box 395
Pretoria 0001

PROFESSOR S. H. VON SOLMS
Department of Computer Science
Rand Afrikaans University
Auckland Park
Johannesburg 2001

PROFESSOR M. H. WILLIAMS
Department of Computer Science
Herriot-Watt University
Edinburgh
Scotland

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R10	$7	£5
Institutions	R15	$14	£10

Circulation and Production Manager

MR. C. S. M. Mueller
Department of Computer Science
University of the Witwatersrand
1 Jan Smuts Avenue
Johannesburg 2001

Quaestiones Informaticae is prepared by the Computer Centre and Central Graphics Unit, and printed by the Central Printing Unit, all of the University of the Witwatersrand, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial Note

I am happy to present the current issue of Quaestiones Informaticae. As indicated in the last issue, Vol.3 No.2, it was necessary to look for a new method to produce the journal. Thanks to the efforts of Prof. Judy Bishop we have found a way to continue at an acceptable cost, and the result is in your hands. Wits University's Computer Centre will do the 'typesetting' using a laser printer. Judy will use Computer Science students to do the proof reading, and Wits will also do the printing. On behalf of myself, the Computer Society of SA and the SA Institute of Computer Scientists, I want to express my appreciation for all their efforts.

Conrad Mueller, of Wits' Computer Science department has agreed to become the circulation manager for QI. He will handle the business side of our publication. A hearty welcome to him!

G. WIECHERS
Editor.

STOP PRESS

QI is now on the official supplementary list of journals considered by the National Department of Education for subsidy purposes.

A Consideration of Formalisms in Computing

R. SHORT
Department of Accounting, University of the Witwatersrand, Johannesburg, 2001 South Africa

1. INTRODUCTION

Formalisms comprise the essential bricks and mortar that we need in order to construct information systems. Not surprisingly therefore the invention and refinement of formalisms has formed and continues to form a significant activity and interest throughout the length and breadth of the computing community from the academic researcher to the work-a-day analyst or programmer. In the light of their significance therefore it would seem to be a worthwhile exercise to endeavour to document something of the process of formalism creation, its motivators and objectives.

2. FORMALISMS

By a formalism is meant a rigorously defined convention for presenting some specified aspect(s) of the world. World is used here in its most catholic sense. The formalisms dealt with in computing normally each comprise a defined set of symbols and a defined set of rules which stipulate what aspect(s) of 'the world' are to be represented by each symbol. An example of such a formalism is the set of standard flowcharting symbols developed by the National Computing Centre in the United Kingdom.[1]

3. THE ROLE OF FORMALISMS IN COMPUTING

Briefly the role of formalisms in computing is to offer a firm handle on complexity. Without doubt any reasonable program let alone a complete information system represent a hotbed of complexity, if not intrigue. It is in fact in our efforts to exorcise the intrigue and mystery generated by the complexity that we bring in formalisms.

Even the most cursory consideration of current computing practice will reveal that formalisms form the cornerstone of successful systems development methods. In fact I would venture that computing practice stumbles most often in just those areas where the necessary formalisms either have not as yet been developed or have not as yet evolved to a level adequate to the tasks for which they were intended.

4. MOTIVATORS OF FORMALISM DEVELOPMENT

Because of the centrality of formalisms to the field of computing and because of its relative youth, computing has proved to be a fertile, attractive and rewarding field of interest for inventors and refiners of formalisms. It is *fertile* because we need appropriate formalisms if we are to practise the computing discipline with any certainty of successful results. It is *attractive* because there is a certain type of mind which delights in formalisms. It is *rewarding* because there is not only creative satisfaction but also social affirmation in creating things which one's community, in this case computer people, finds useful.

Evidence of the effective strength of these motivators to formalism creation can be seen right from the earliest days of computing. The development of programming languages begun then, and it is an activity which continues to this day.

In addition because people are not very good at translating higher level formalisms into a more detailed formalisms needed at hardware level a significant adjunct to formalism creation has been the development of technology to handle the translation task. Translators/assemblers, compilers and program generators are examples of this technology.

The mind which delights in formalisms fuels the drive toward this particular technology. This is illustrated by what a keen programmer, name unknown to me, is reputed to have once said: "I like programming but I like even more writing programs to help me program."

5. DEVELOPING FORMALISMS

5.1 A PRECIPITATING FACTOR

Anybody who has worked for any length of time in the formal aspects of computing, such as systems analysis and design or software design and programming, will probably know that feeling of intellectual discomfort and frustration that arises when the intellectual tools at our disposal do not seem to be able to quite cope with the things we are trying to master. No doubt these are the feelings which frequently start us on the path to refining an existing formalism or creating a new one if need be.

5.2 THE PROCEDURE FOLLOWED

I am not aware of any documented procedures for creating formalisms. However, there are clearly certain general things that must be considered. Firstly decisions need to be made about:

a. which of the aspects/features of the thing to be represented do we regard as being necessary to represent with the formalism;
b. what is the hierarchical structure within which these aspects/features reside;
c. which of the inter-aspect/inter-feature, relationships do we regard as being necessary to represent with the formalism.

Once having decided these things we can then set about determining exactly how the individual items are going to be represented in the formalism.

As the major purpose of any formalism is to serve as a rigorous vehicle for human communication, the communicative abilities, e.g. visual clarity etc. of any representations chosen for inclusion in a formalism must be carefully weighed before the adoption of the representation.

For further elucidation of the points just covered let me turn to a consideration of some aspects of existing formalisms intended for program design. Other formalisms could equally well have been chosen.

5.3 A CONSIDERATION OF SOME ASPECTS OR PROGRAM FORMALISMS

What are the aspects /features of a program that we wish/need to be able to represent with as a formalism intended for use in program design?

Let us look just two of these aspects/features of programs. They are:

a. component parts;
b. flow of control from one component part to another.

REPRESENTING COMPONENT PARTS

Let us look at the representation of component parts. There are a number of aspects of component parts that we may want to represent but let us look at just two of them for the purpose of illustration.

a. the division (or separation) between a component and everything else;
b. the relationship of a component to the 'whole ' of which it forms part.

Now, in what we can perhaps label as the flowcharting days of programming, very little attention was paid to point (b) above and as a consequence its representation was not treated as important. As far as point (a) is concerned, in flowcharting days it was represented in two ways. Firstly the general conventions for divisions of good written language were assumed to continue to hold. Thus a sentence is a component of a paragraph of text and so on. Secondly the graphical convention of drawing an enclosing boundary line to mark out a component was widely used.

As programming matured the down playing of point (b) was recognised as a major deficiency and structure charting was developed to enable point (b) to explicitly represented.

In structure charting the representation of point (b) is achieved through a combination of two conventions. Lines are drawn to link 'wholes' to the representations of their individual component parts whilst the representations of the 'wholes' are placed in normal written text precedence positions in relation to their component parts, that is above them on the page. Thus, structure charting can be seen as formalism refinement to cater for an aspect of programs that was ignored by flowcharting.

REPRESENTING FLOW OF CONTROL FROM COMPONENT TO COMPONENT

In the flowcharting days of programming, flow of control, it would seem, was seen to be much like the spirit: it bloweth where it listeth. Thus one of the features of flow control that was regarded as needing to be represented was natural sequence. I will not be considering here that other necessary aspect of flow control, conditional sequence switching. For representing natural sequence in flow of control two conventions were used.

Firstly normal written text sequence conventions were taken as a given. That is a flow of control sequence was always implied that embraced left to right within top to bottom of the page, unless it was clear that some other sequence was intended or needed.

This convention was and is not sufficient however, for the situation where a page contains line bounded components. Thus a second explicit convention was introduced for this representational problem: component connecting lines with arrow heads to show the direction of the flow of control from one boxed component to the next. A weakness arising from the flexibility of this representation was in due course perceived, however.

Experience taught programmers that the mastery of complexity was a very important aspect indeed of successful programming and if the necessary transfers of control between program components took no cognisance of, and in no way reflected the 'wholes' and 'wholes of wholes' of which the components were part then complexity was in no way mastered, it was multiplied.

It was then realised that complexity is controlled if natural sequence transfers of control are designed to take place only between the adjacent components of a single whole.[2] For example (fig 1) if a transfer of control is needed from component I to N which are both members of two different wholes F and H not themselves components of the same whole then control must be transferred up the hierarchy until the transfer can take place between the two adjacent components B and C which are members of the same whole A, and then passed back down the hierarchy to N.

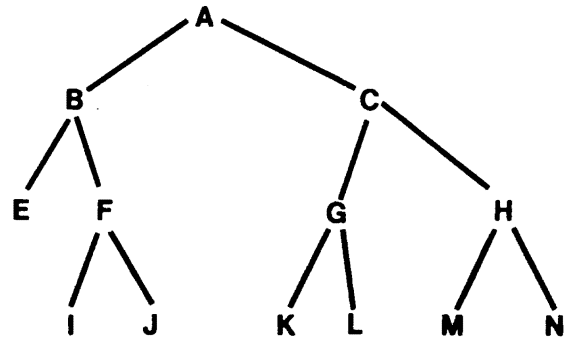

Figure 1. A COMPONENT CHART

Thus flow of control is firmly and explicitly disciplined in the structure charting formalism as opposed to the situation in the flow-charting formalism where it is , so to speak, free to roam at will.

This new formalism no doubt sprung out of frustrations with the complexity consequences of the earlier approach to flow of control as evidenced in flow-charting.

6. CONCLUSION

In this article I have only used a few particular aspects of program design formalisms to illustrate my general points about the creation process for formalisms in computing.

However, I hope an awareness of the process by which computing formalisms come into being will help computing professionals to cope constructively with feelings of dissatisfaction over an existing formalism by enabling them to systematically:

a. evaluate the formalism
b. improve it or create a new and better one.

Also, I have not expanded at all upon the issues and opportunities arising from the need to provide computing mechanisms for converting one formalism to another. I leave consideration of these problems and the undoubted opportunities to the reader.

REFERENCES

1. *Introducing Systems Analysis & Design* **Vol 1**, Pub. The National Computing Centre, Manchester, U.K.

2. *JSP A practical Method of Program Design*, Leif Ingevaldsson, Pub. Chartwell-Bratt Ltd. England

NOTES FOR CONTRIBUTORS

The purpose of this journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review articles, exploratory articles of general interest to readers of the Journal. The preferred languages of the Journal will be the congress languages of IFIP although papers in other languages will not be precluded.

Manuscripts should be submitted in triplicate to:
Prof. G. Wiechers at:
INFOPLAN
Private Bag 3002
Monument Park 0105

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. The original ribbon copy of the typed manuscript should be submitted. Authors should write concisely.

The first page should include the article title (which should be brief), the author's name, and the affiliation and address. Each paper must be accompanied by an abstract less than 200 words which will be printed immediately below the title at the beginning of the paper, together with an appropriate key word list and a list of relevant Computing Review categories.

Tables and figures

Illustrations and tables should not be included in the text, although the author should indicate the desired location of each in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Illustrations should also be supplied on separate sheets, and each should be clearly identified on the back in pencil with the Author's name and figure number. Original line drawings (not photoprints) should be submitted and should include all relevant details. Drawings etc., should be submitted and should include all relevant details. Drawings etc., should be about twice the final size required and lettering must be clear and "open" and sufficiently large to permit the necessary reduction of size in block-making.

Where photographs are submitted, glossy bromide prints are required. If words or numbers are to appear on a photograph, two prints should be sent, the lettering being clearly indicated on one print only. Computer programs or output should be given on clear original printouts and preferably not on lined paper so that they can be reproduced photographically.

Figure legends should be typed on a separate sheet and placed at the end of the manuscript.

Symbols

Mathematical and other symbols may be either handwritten or typewritten. Greek letters and unusual symbols should be identified in the margin. Distinction should be made between capital and lower case letters; between the letter O and zero; between the letter I, the number one and prime; between K and kappa.

References

References should be listed at the end of the manuscript in alphabetical order of author's name, and cited in the text by number in square brackets. Journal references should be arranged thus:

1. ASHCROFT,E. and MANNA,Z.(1972).The Translation of 'GOTO' Programs to 'WHILE' Programs, in *Proceedings of IFIP Congress 71*,North-Holland, Amsterdam, 250-255.

2. BOHM,C. and JACOPINI,G.(1966).Flow Diagrams, Turing Machines and Languages with only Two Formation Rules, *Comm.ACM*, 9,366-371.

3. GINSBURG,S.(1966). *Mathematical Theory of context-free Languages*, McGraw Hill, New York.

Proofs and reprints

Galley proofs will be sent to the author to ensure that the papers have been been correctly set up in type and not for the addition of new material or amendment of texts. Excessive alterations may have to be disallowed or the cost charged to the author. Corrected galley proofs, together with the original typescript, must be returned to the editor within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only original papers will be accepted, and copyright in published papers will be vested in the publisher.

Letters

A section of "Letters to the Editor" (each limited to about 500 words) will provide a forum for discussion of recent problems.

Hierdie notas is ook in Afrikaans verkrygbaar.