4de th

SUID-AFRIKAANSE REKENAARSIMPOSIUM
SOUTH AFRICAN COMPUTER SYMPOSIUM

HOLIDAY INN PRETORIA
JULIE 1 – 3 JULY 1987

Proceedings

of the

4th South African Computer Symposium

Holiday Inn, Pretoria

1 – 3 July 1987

edited by

Pieter Kritzinger

Computer Science Department
University of Cape Town

PREFACE

Computer science is an emerging discipline which is having difficulty in being recognised as a worthy member of the sciences. I will paraphrase John Hopcroft, co-winner of the 1986 Turing Award, when, during a recent interview, he said that the primary reason for the lack of recognition, is the age of our researchers. Probably not one of the researchers who presented their work at this symposium is older than 45. I know of no computer scientist in South Africa who is in a position where (s)he can affect funding priorities. As far as I know we have no representation on any of the committees of the Foundation for Research Development and for our Afrikaans speaking fraternity, none who is a member of the *Akademie vir Wetenskap en Kuns*. It will take time and conscious effort to establish our presence. The same is true of course for our universities. Again, with one exception, I know of no dean of a science faculty, vice-principal or principal who is a computer scientist. We consequently spend an enormous amount of time trying to explain the needs of computer science and its difficulties. I believe this symposium is a further step towards accreditation by our peers and superiors from the other sciences.

The total number of papers submitted to the Programme Committee for consideration was 34. Each paper was reviewed by three persons knowledgeable in the field it represents. Of those submitted, 23 were finally selected for inclusion in the symposium. As a result the overall quality of the papers is high and as a computer science community in Africa we can be justly proud of the final programme.

This is the fourth in the series of South African computer symposia. This year the symposium is sponsored by the Computer Society of South Africa (CSSA), the South African Institute for Computer Scientists and the local IFIP Committee. The executive director of the CSSA and his staff deserve warm thanks for handling the organisation as well as they have, while the Organising Committee provided Derrick and I with very valuable advice.

Finally I would like to express my sincere appreciation to the authors, to the members of the Programme Committee and particularly the reviewers. Without the kind cooperation of everyone, this symposium would not have taken place.

Pieter Kritzinger
July 1987.

SYMPOSIUM CHAIRMAN: PS Kritzinger, University of Cape Town.

SYMPOSIUM CO-CHAIRMAN: D Kourie, University of Pretoria.

MEMBERS OF THE PROGRAMME COMMITTEE

Judy Bishop, Witwatersrand University

Chris Bornman, UNISA

Hannes de Beer, Potchefstroom University

Gideon de Kock, Port Elizabeth University

Jaap Kies, Western Cape University

Derrick Kourie, Pretoria University

Pieter Kritzinger, Cape Town University

Tony Krzesinski, Stellenbosch University

Michael Laidlaw, Durban Westville University

Peter Lay, Cape Town University

Ken MacGregor, Cape Town University

Theo McDonald, Orange Free State University

Jan Oosthuizen, University of the North

Dennis Riordan, Rhodes University

Alan Sartori-Angus, Natal University

John Shochot, Witwatersrand University

Theuns Smith, Rand Afrikaans University

Trevor Turton, ISM (Pty) Ltd

Gerrit Wiechers, Infoplan.

LIST OF REVIEWERS

BERMAN Sonia
BISHOP Judy
BORNMAN Chris
CAREY Chris
CHERENACK Paul
DE BEER Hannes
DE VILLIERS Pieter
GORRINGE Pen
KIES Jaap
KOURIE Derrick
KRITZINGER Pieter
KRZESINSKI Tony
LAIDLAW Michael
LAY Peter
MacGREGOR Ken
MATTISON Keith
McDONALD Theo
RENNHACKKAMP Martin
RIORDAN Denis
SATORI-ANGUS Alan
SCHOCHOT John
SMITH Theuns
TURTON Trevor
VAN DEN HEEVER Roelf
VAN ROOYEN Hester
VON SOLMS Basie
VOS Koos

TABLE OF CONTENTS

Computer Languages III

Tutorial

DAY 1

07h30 Registration and Coffee.

08h45 Welcoming address, President of the South African Institute of Computer Scientists, Dr. G. Wiechers.

09h00 Invited Lecture.
Professor D. Teichrow, University of Michigan.
Software Engineering, ... What Can We Expect in the Future.

10h00 COFFEE

Computer Languages I. Chairman: G. Wiechers.

10h15 S. Berman, University of Cape Town.
SPS-Algol: Semantic Constructs for a Persistent Programming Language.

10h50 A. Watson, University of the Witwatersrand.
Petri Net Topologies for a Specification Language.

11h25 R. Mori, University of Cape Town.
Towards a Programming Environment Standard in USP.

11h50 J. Bishop, University of the Witwatersrand.
ADA for Multiprocessors: Some Problems and Solutions.

12h30 LUNCH

	Computer Graphics. Chairman: D. Kourie	**Operating Systems.** Chairman: K. MacGregor.
14h00	C. F. Scheepers, CSIR. *Polygon Shading on Vector Type Devices.*	M. Morris, UNISA. *The Development of a Fault Tolerant System for a Real-time Environment.*
14h35	P. Gorringe, CSIR. *Hidden Surface Elimination in Raster Graphics Using Visigrams.*	B. H. Venter, CSIR. *A New General-purpose Operating System.*
15h15	COFFEE	COFFEE
	Database Systems I. Chairman: B. von Solms.	**Computer Languages II.** Chairman: J. Bishop.
15h30	M.E. Orlowska, UNISA. *On Syntax and Semantics Related to Incomplete Information Databases.*	E.M. Ehlers and B. von Solms, Randse Afrikaanse Universiteit. *The Representation of Chemical Structures by Random Context Structure Grammars.*
16h05	M.H. Rennhackkamp, Stellenbosch University. *Modelling Distributed Database Concurrency Control Overheads*	W. van Biljon, CSIR. *A Generalised Expression Structure.*

18h00 Cocktail Party in Cullinan Room A.

DAY 2

08h30 Keynote Address by Profesor Niklaus Wirth, Swiss Federal Institute for Technology, Zurich.
An Extensible System and a Programming Tool for Workstation Computers.

Computer Networks and Protocols I. Chairman: P.S. Kritzinger.

09h30 A.E. Krzesinski, University of Stellenbosch.
An Approximate Solution Method for Multiclass Queueing Networks with State Dependent Routing and Window Flow Control.

10h05 J. Punt, University of Cape Town.
A Protocol Validation System.

10h30 COFFEE

Computer Networks and Protocols II. Chairman: R. van der Heever.

11h00 P.S. Kritzinger, University of Cape Town.
Protocol Performance using Image Protocols.

11h35 Invited Lecture by Professor G. Lasker, University of Windsor, Ontario.
The Relationship of Natural and Artificial Intelligence.

12h30 LUNCH

	Artificial Intelligence. Chairman: G. Lasker.	**Information Systems.** Chairman: D. Teichrow.
14h00	A. Cooper, CSIR *A Data Structure for Exchanging Geographic Information.*	L. du Plessis and C. Bornman, UNISA. *The ELSIM Language: an FSM-based Language for the ELSIM SEE.*
14h35	A. I. Newcombe, University of Cape Town and R. Rada, National Library of Medicine, Maryland. *Strategies for Automatic Indexing and Thesaurus Building.*	J. Mende, University of the Witwatersrand. *Three Packaging Rules for Information System Design.*
15h15	COFFEE	COFFEE
	Database Systems II. Chairman: C. Bornman.	**Computer Languages III.** Chairman: N. Wirth.
15h30	M.J. Philips, CSIR. *An Approach to Direct End-user Usage of Mutiple Databases.*	W. van Biljon, CSIR. *Experience with a Pattern-matching Code Generator.*
16h05	S. Berman, University of Cape Town. *A Semantic Data Model Approach to Logical Data Independence.*	C. Mueller, University of the Witwatersrand. *Set-oriented Functional Style of Programming.*

16h45 Open Forum with professors G. Lasker, D. Teichrow and N. Wirth.
Moderator: Dr. D. Jacobson.

19h30 Symposium Banquet in Cullinan Room.
Guest speaker, Dr. D. Jacobson, Group Executive: Technology, Allied Technologies Limited.

DAY 3

08h00 Registration (Tutorial only).

08h30 Tutorial.
The Tutorial will be given by professor Niklaus Wirth, Division of Computer Science, Swiss Federal Institute of Technology, Zurich.

The use of Modula-2 in Software Engineering.
Topics to be covererd include:

What is Software Engineering?
Data types and structures.
Modularization and information hiding.
Definition and implementation parts.
Separate compilation with type checking.
Facilities to express concurrency.
Pompous programming style.
What could be excluded?

12h15 **Close of Symposium.**

12h30 LUNCH

THREE PACKAGING RULES FOR INFORMATION SYSTEM DESIGN

J. Mende

Department of Accounting
University of the Witwatersrand
WITS 2050

ABSTRACT

After identifying the processing functions required in a computer based information system, the designer needs to combine them into an optimal set of load units. Some "packaging" arrangements yield a better system than others, depending upon characteristics of the data collected from external sources and the data extracted for external users. An effective and technically efficient system satisfies three rules.

1. If two user data types are needed at different times, the corresponding extract functions should be separated in different load units.

2. If source data predates the user data derived from it, the corresponding collect and extract functions should be separated in different load units.

3. If two source data types are available at different frequencies, one being less frequent than the user data derived from it, the corresponding collect functions should be separated in different load units.

Business, government and other organisations employ a majority of the computers in existence today to transform raw data into useful information. The transformation process usually involves a large number of distinct processing "functions" (4) such as validation, updating, sorting, retrieval and accumulation. Computer memories today are often so large that all the functions necessary to accomplish a complex transformation can be incorporated in one single program. However, in many cases that arrangement wastes computing resources. So instead those functions are incorporated into several smaller "load units" - programs, overlays, subroutines, etc. Accordingly, in developing a new computer based information system the designer has to decide how to divide the set of all necessary functions into separate load units. However, this "packaging decision" is not always easy. The set of all functions can usually be partitioned in many alternative ways: so finding the optimal arrangement represents a difficult problem. To help him solve the problem, the designer needs formal packaging rules.

A parallel paper (11) demonstrates that the typical rule should consist of two parts - a condition and a comparison. The condition identifies a particular kind of design situation. The comparison predicts the better of two alternative functional arrangements in terms of some criterion of success. Several kinds of conditions, functional arrangements and success criteria are distinguishable. That means many different types of rules are needed.

Yourdon and Constantine (17) have established the most comprehensive set of packaging rules currently available in the Information Systems literature (2,3,12,13,14). Those rules are concerned with one of three possible success criteria: "technical efficiency" (10). They compare two kinds of functional arrangement:

- "associative", i.e. functions combined in the same load unit , and
- "dissociative" i.e. functions separated in different load units.

They address situations in which functions are connected, sequentially incompatible, once-off and run-optional:

- Rule A. Include in the same load unit functions connected by iterated reference.
- Rule B. Include in the same load unit functions with high volume of access on connecting references.
- Rule C. Include in the same load unit functions with high frequency of access on connecting references.
- Rule D. Include in the same load unit as the superordinate any functions with short interval of time between activation.

- Rule E. Put into a separate load unit any optional function.

- Rule F. Put into a separate load unit any function used only once.

- Rule G. Put functions applied on input and output sides of a sort into separate load units.

However, certain situations occur which are not explicitly mentioned in these rules. In particular, the 1974 design technique of Waters (15,16) suggests that a designer often encounters functions that receive inputs supplied by external data sources, or produce outputs consumed by external information users, and that these should normally be separate. The same distinction was re-iterated in 1982 (5) and 1983 (1). The present paper follows up the Waters clue to establish three new rules which may be added to the Yourdon-Constantine set. Following the methodological guidelines developed in three earlier papers (7,8,9) these rules will be derived logically from three underlying premises about information systems.

The first premise concerns system success. An information system inputs resources such as labour, hardware, software and raw data from its environment; in exchange it outputs processed data needed by the environment. The system is "successful" if the value v of its outputs exceeds the cost c of its inputs, i.e. the ratio v/c is maximal. It has been shown (10) that this ratio is the product of three independent success criteria:

- effectiveness, i.e. how well do outputs satisfy environmental needs?

- economic efficiency, i.e. how cheap is the resource mix?

- technical efficiency, i.e. are resources wasted?

A second premise distinguishes between "load unit" and "function". In order to transform raw data into information, a computer typically performs many individual operations such as reading, writing, addition, etc. These operations are initiated by instructions situated in some rapidly accessible device which is defined here as the "program memory". To get those instructions into the program memory, the computer normally loads them from some kind of external library. For the sake of technical efficiency, the loader transfers several instructions at a time, so that execution only begins after an entire group of instructions has been loaded. Such a group is defined as a load unit (17). Packaging is only feasible if every function fits into some load unit in its entirety. Therefore a function can be defined as a subset of a load unit which accomplishes some subtask of a system's overall transformation task.

The third premise distinguishes between "collect" and "extract" functions. An I.S. provides outputs needed by its environment: consequently the system must contain functions which produce that output. Similarly, an I.S. receives inputs supplied by its environment, and therefore the system must include functions which accept that input. As the terms "input" and "output" denote many different things, the two functions will be defined more precisely:

- a collect function inputs source data from its environment

- an extract function outputs user data to its environment.

The term "source data" includes data received from the organisation, its customers and suppliers, as well as data received directly from other information systems. The term "user data" includes information provided to the organisation, its customers and suppliers, as well as data transferred directly to other information systems.

The premises reflect features normally found in computer based information systems today. They are not "universal" in the sense that they are true of every single information system in existence, but there are so few exceptions that they represent the "typical" system. In contrast, the remainder of this paper examines situations which are commonly encountered, but not so often that they can be described as "typical".

In the first situation, several functions are executed at inherently different times. For example, in a batch-processing Debtors system, the statements print function might be executed once per month and the validate function once per week. In a real-time Debtors system, a validation function might collect sales data in real-time; an update function might collect a file of cash receipts once a day; a print function might produce statements once a month, and an enquiry function might extract individual debtors accounts on demand. Such functions are "temporally independent". Consider two such functions, F and G. Suppose they were both included in the same load unit. Then, whenever F needs to be executed, both F and G would be loaded into the program memory - but G would not be needed. Similarly, whenever G needs to be executed, both F and G would be loaded - but now F would not be needed. In both cases loading time and program memory would be wasted. Therefore technical efficiency demands a dissociative arrangement, and so the Yourdon-Constantine Rule E can be re-stated as

> Rule 0: if two functions are temporally independent, a packaging arrangement which separates them is more technically-efficient than an arrangement which combines them in the same load unit.

The second situation involves an information system environment which demands different kinds of user data at different times. For example, users of a Debtors system might require real-time answers to ad-hoc enquiries on the one hand, and monthly statements on the other. A Sales Orders system might be required to produce hourly picking lists, as well as a daily transfer file of sales data to the Debtors system. Users of a Stores system might need daily stock reorder lists, and real-time answers to stock-level enquiries. In these and many other systems the various user data types are temporally independent: each is needed at an inherently different time. Suppose such a system contains an extract function E which produces user data type U. Then there are three alternatives.

- E may be executed well after U is needed. In this case U will be late and therefore the system will be ineffective.

- E may be executed well before U is needed. In this case U may be incomplete, as source data collected in the interval u to x cannot be reflected in U; so again U will be ineffective.

- E may be executed close to the time U is needed. This alternative avoids the previous drawbacks: so U is maximally effective.

Next, consider two extract functions E_1 and E_2 whose user data U_1 and U_2 are needed at different times, u_1 and u_2. If E_1 and E_2 are executed at the same time, say x, then there are three timing alternatives:

- x may be close to u_1, in which case U_2 will be ineffective

- x may be close to u_2, in which case U_1 will be ineffective

- x is close to neither, so both U_1 and U_2 will be ineffective.

However, if E_1 were executed near u_1 and E_2 near u_2, then both U_1 and U_2 will be effective. Therefore Rule 0 leads to ...

> Rule 1: if two user data types are temporally independent, a packaging arrangement which separates the corresponding extract functions is more effective and technically efficient than an arrangement which combines them in the same load unit.

The third situation involves an environment which needs user data based on source data generated a relatively long time ago. For example, users of a Debtors system might need statements which summarise sales and cash transactions that occurred at the beginning and middle of the month. Users of a Budgeting system might need variance analyses based on plans made up to a year ago. Users of a Sales Forecasting system might require forecasts based on invoices generated during the past three to five years. In these and many other systems source data "predates" user data. Consider a collect function C and an extract function E, where the source data S received by C predates the user data U produced by E. As shown for

temporally independent user data, the system can only be effective if E is executed near u, the time at which U is needed. U cannot be produced unless S has previously been collected, so C must be executed at some time x prior to u. That time may be close to u or well before u. Suppose x is close to u. Then as there is always some chance that source data may contain errors which will be rejected by C, and those errors are unlikely to be corrected before E is executed, there is a finite probability that U will be incomplete. After the system has been used a few times, that probability becomes a certainty, and the system would be ineffective. So C should be executed well before E, and therefore Rule 0 leads to .....

> Rule 2: if source data predates user data, a packaging arrangement which separates the corresponding collect and extract functions is more effective and technically efficient than an arrangement which combines them in the same load unit.

The last situation involves an environment which supplies different kinds of source data at different frequencies. For example, in a Debtors system sales data might arrive every few minutes from a terminal; a transfer file of receipts data might be available once per month; statements might be needed at month-end, and customer accounts might have to be displayed at any time. In a Stores system, material movements data may be generated continuously; a transfer file of purchase orders may be available once per day; a stock reorder list might be needed once per day, and stock levels might have to be displayed at any time. In a Budgeting system, plan data might be generated annually; performance data might be available weekly, and variance reports might be needed monthly. Consider an extract function E and two collect functions C_1 and C_2 in such a system. Source data are generated at frequencies s_1 and s_2; the user data are needed at frequency u. Suppose

$$s_1 \geqslant u \quad \text{but} \quad s_2 < u.$$

As in Rule 1, effectiveness demands that E should be executed at frequency u. Now if C_1 were executed less frequently than E, then E would not always have data available to it: so effectiveness also demands that C_1 be executed at frequency $c_1 \geqslant u$. However, a C_2 execution frequency $c_2 > s_2$ is futile: so $c_2 \leqslant s_2$. As

$$s_2 < u \text{ and } u \leqslant c_1$$

that means $c_2 < c_1$. Therefore C_1 and C_2 should be executed at different times. So Rule 0 leads to ....

> Rule 3: if two source data types are available at different frequencies, one being less frequent than the user data type derived from it, then a packaging arrangement which separates the corresponding collect functions is more effective and technically efficient than an arrangement which combines them in the same load unit.

The Yourdon-Constantine packaging rules are aimed at technical efficiency and primarily address intra system situations: connected modules, processing sequence and internal frequency. (Only Rule E can be applied in situations involving a system's environment). In contrast, rules 1 - 3 are aimed at effectiveness and primarily address inter-system situations: interactions between an information system and a business system or another information system. Therefore they should serve as significant extensions to the internally-oriented Yourdon-Constantine set.

The way the new rules have been established is also significant. The validity of the Yourdon-Constantine rules rests on their intuitive appeal. They "make sense" in the case studies presented by the authors; and an experienced designer can recall many additional instances in which they are consistent with his own packaging decisions. In contrast, the present paper presents formal proofs. It shows that Information Systems principles can be derived by chains of logical reasoning from underlying patterns. This suggests that proofs can be also constructed for our other unsubstantiated "rules of thumb", so that the subject Information Systems may well become more scientific one day (6).

REFERENCES

1. Clifton, M.D. (1973). Business Data Systems. 2nd ed. Prentice-Hall International, London, p229.

2. Gomaa, H. (1984). A software design method for real-time systems. Comm. ACM, 27, 938 - 949.

3. Jackson, M.A. (1983). System Development. Prentice-Hall International, London.

4. Jensen, R.W. and Tonies, C.C. (1979). Software Engineering. Prentice-Hall,Englewood Cliffs,New Jersey, p120.

5. Mende, J. (1982). Teach systems the deductive way. The Commerce Teacher, 14, 43 - 45.

6. Mende, J. (1986). Research Directions in Information Systems. Quaestiones Informaticae, 4(1), 1 - 4.

7. Mende, J. (1986). Laws and Techniques of Information Systems. Quaestiones Informaticae, 4(3), 1 - 6.

8. Mende, J. (1987). A Structural Model of Information Systems Theory. To appear in Quaestiones Informaticae.

9. Mende, J. (1987). A Methodology for Research on Information Systems. Working paper, University of the Witwatersrand.

10. Mende, J. (1987). Three objectives of information system design. SACLA Conference, Pretoria.

11. Mende, J. (1987). A classification of information systems decomposition rules. SACLA conference, Pretoria.

12. Myers, G.J. (1978). Composite Structured Design. Van Nostrand Reinhold, New York, p.6.

13. Randell, B (1986). System Design and Structuring. The Computer Journal, 29, 300 - 306.

14. Stevens, W.P. (1981). Using Structured Design. Wiley-Interscience, New York.

15. Waters, S.J. (1974). Methodology of computer systems design. The Computer Journal, 17, 17 - 24.

16. Waters, S.J. (1974). Introduction to Computer Systems Design. NCC Publications, Manchester.

17. Yourdon, E. & Constantine L. (1979). Structured Design. Prentice-Hall, Englewood Cliffs, New Jersey, p.276 - 289.