ISSN 0254–2757

QI QUÆSTIONES INFORMATICÆ

Volume 5 • Number 3 December 1987

M.E. Orlowska	**Common Approach to Some Informational Systems**	**1**
S.P. Byron-Moore	**A Program Development Environment for Microcomputers**	**13**
N.C.K. Phillips S.W. Postma	**Pointers as a Data Type**	**21**
P.J.S. Bruwer J.J. Groenewald	**A Model to Evaluate the Success of Information Centres in Organizations**	**24**
J. Mende	**Three Packaging Rules for Information System Design**	**32**
T. D. Crossman	**A Comparison of Academic and Practitioner Perceptions of the Changing Role of the Systems Analyst: an Empiral Study**	**36**
P.J.S. Bruwer	**Strategic Planning Models for Information Systems**	**44**
S.H. von Solms	**Generating Relations Using Formal Grammars**	**51**
A.L. du Plessis C.H. Bornman	**The ELSIM Language: an FSM-Based Language for ELSIM SEE**	**67**
	BOOK REVIEW	**56**
	CONFERENCE ABSTRACTS	**57**

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

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

QUÆSTIONES INFORMATICÆ

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-Africa en van die Suid-Afrikaanse Instituut van Rekenaarwetenskaplikes

Subscriptions

Annual subscription are as follows:

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

Computer Society of South Africa
Box 1714 Halfway House

Quæstiones Informaticæ is prepared by the Computer Science Department of the University of the Witwatersrand and printed by Printed Matter, for the Computer Society of South Africa and the South African Institute of Computer Scientists.

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.

1. PACKAGING RULES

Business, government and other organizations 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: (a) *associative*, i.e. functions combined in the same load units, and (b) *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

questiones informaticæ, 5, 3, pp 32-35, issn 0254-2757

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.

2. UNDERLYING PREMISES

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: (a) *effectiveness*, i.e. how well do outputs satisfy environmental needs? (b) *ecomonic efficiency*, i.e. how cheap is the resource mix? (c) *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:

i) a collect function inputs source data from its environment, and
ii) an extract function outputs user data to its environment.

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

3. THREE NEW RULES

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 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: (a) x may be close to u_1, in which case U_2 will be ineffective; (b) x may be close to u_2, in which case U_1 will be ineffective; (c) 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 month-end statements which summarize 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 $s_1 \geq u$ but $s_2 < u$. 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 $c_1 > u$. However, a C_2 execution frequency $c_2 > s$ is futile: so $c_2 \leq s_2$. As $s_2 < u$ and $u < c_1$ 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.

4. IMPLICATIONS

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, 229.
2. Gomaa, H. [1984]. A software design method for real-time systems. *Communications of the 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, 120.
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.
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, 276 -289.

NOTES FOR CONTRIBUTORS

The purpose of the journal will be to publish original papers in any field of computing. Papers submitted may be research articles, review artilces and 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
INFOPLAN
Private Bag 3002
Monument Park 0105
South Africa

Form of manuscript

Manuscripts should be in double-space typing on one side only of sheets of A4 size with wide margins. Manuscripts produced using the Apple Macintosh will be welcomed. Authors should write concisely.

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

Tables and figures

Tables and figures should not be included in the text, although tables and figures should be referred to in the printed text. Tables should be typed on separate sheets and should be numbered consecutively and titled.

Figures should also be supplied on separate sheets, and each should be clearly identified on the back in pencil and the authors name and figure number. Original line drawings (not photocopies) should be submitted and should include all the relevant details. Drawings etc., should be submitted and should include all relevant details. Photographs as illustrations should be avoided if possible. If this cannot be avoided, glossy bromide prints are required.

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 alphabetic order of the author's name, and cited in the text in square brackets. Journal references should be arranged thus:

1. Ashcroft E. and Manna Z., The Translation of 'GOTO' Programs to 'WHILE' programs., *Proceedings of IFIP Congress 71*, North-Holland, Amsterdam, 250-255, 1972.
2. Bohm C. and Jacopini G., Flow Diagrams, Turing Machines and Languages with only Two Formation Rules., *Comm. ACM*, **9**, 366-371, 1966.
3. Ginsburg S., Mathematical Theory of Context-free Languages, McGraw Hill, NewYork, 1966.

Proofs

Proofs will be sent to the author to ensure that the papers have been correctly typeset and *not* for the addition of new material or major amendment to the texts. Excessive alterations may be disallowed. Corrected proofs must be returned to the production manager within three days to minimize the risk of the author's contribution having to be held over to a later issue.

Only orginal 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.