J. M. Bishop
8/7/84

Quaestiones Informaticae

Vol. 3 No. 2 July, 1984

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
Department of Computer Science and Information Systems
University of South Africa
P.O. Box 392, Pretoria 0001

Subscriptions

Annual subscriptions are as follows:

	SA	US	UK
Individuals	R6	$7	£3,0
Institutions	R12	$14	£6,0

Circulation manager
Mr E Anderssen
Department of Computer Science
Rand Afrikaanse Universiteit
P O Box 524
Johannesburg 2000
Tel.: (011) 726-5000

Quaestiones Informaticae is prepared for publication by Thomson Publications South Africa (Pty) Ltd for the Computer Society of South Africa and the South African Institute of Computer Scientists.

Editorial Note

Regrettably this is the last issue of *Questiones Informatica* to appear in its current format. There are a number of reasons for terminating the production of *QI* in printed form. Firstly, the cost of publication has tripled over the last couple of years. Secondly, the interest in the journal has dwindled to the point where one may question the need for a South Arican publication on the more academic aspects of computing: the readership has not expanded but, more seriously, despite many appeals no contributions are submitted. At present I have only one paper in the pipeline. It appears that the only supply of papers comes from the two-yearly Computer Science Symposium, and this is not sufficient to justify the expense of publishing *QI* in its present form.

Nevertheless, both the Computer Society and the Institute of Computer Scientists feel that a vehicle for publishing the results of research in the field of information technology is required for this country. We propose to continue *QI* in a format similar to that used by *Questions Informatica*. In other words we shall use a photo-reproduction process and a printed cover to continue the publication of *QI*, but at greatly reduced costs. This puts the onus of delivering reproducible copy of the authors, but relieves the printers from the problems experienced with unusual symbols, diagrams, computer printouts, etc.

The journal will continue to accept only papers which have been referred. In fact, we shall try to satisfy all the requirements for being regarded as a publication acceptable under the rules for obtaining university subsidies. It is also intended to guarantee quick publication, say not more than three months after acceptance of a paper. However, the journal can only continue to exist if it receives enough contributions of sufficiently high standard.

Again, I would like to appeal to all engaged in research and development in computer related areas, to consider publishing your results in *QI*.

Finally, it is my pleasure to thank Dick White, of Thomson Publications, for all he has done to produce this journal in its current form.

G WIECHERS,
Editor

Syllabi for Computer Science as a Scientific Discipline

Stef W. Postma
University of South Africa

Abstract
The paper is concerned with computer science as a science and the training required by its scientific practitioners-to-be. The background and objectives for a curriculum design are briefly sketched, and a taxonomy is given for computer science. The syllabi for the undergraduate courses and an honours course are described and justified. The masters degree is briefly considered. To clarify the undergraduate courses the mathematical pre- and co-requisites are touched upon.

Introduction

A number of things have happened over the last seven or so years, some of them continuing, that prompt a reappraisal of computer science and its teaching at universities. The following are the more important:

1. The Japanese 'fifth generation' initiative.
2. The response of western civilization — e.g. The Alvey report, and American initiatives.
3. The establishment of a core of knowledge in computer science.
4. The emergence, in the seventies, of tertiary educational establishments between universities and technical colleges. These are the Colleges of Advanced Education in Australia, the Technikons in South Africa.

There are also some continuing processes which should be taken into account:

1. The shortage of skilled personnel.
2. The availability of cheap hardware.
3. The automatization of skilled work.

The problems arising from the above are not discussed in this paper, neither does this paper contain a complete solution, but it addresses one aspect of the problems and proposes a start towards a solution to these: In order to create work for people and to exploit the capabilities of modern computers we need a skilled group of people — computer scientists — and their training is discussed below.

An Input-Output Diagram

In setting up a curriculum the users, i.e. students, should be considered as input to the teaching/training process, the graduates and indeed all who take a course constitute the output. Such a diagram is given in fig. 1.

In the diagram the figures given show the typical percentages of students who drop out, pass but do not continue etc. At the masters level a spectrum of results are considered and the MBA is included to clarify this spectrum, it is not proposed that the MBA be offered in a Computer Science department, and it may be doubtful whether the Masters in Information Systems should be offered in the C.S. department or in a Business Economics setting. A similar spectrum holds with respect to a masters degree in electrical engineering, but this is not shown.

If we take a typical first year group through the diagram, them the following results (i.e. graduates etc.) are produced:

Intake at first year level:	200
33 % pass and enter 2nd year:	67
Most pass the BSc:	60
Typical honours class (33 %):	20
At the masters level:	10

9 of the 10 do an essentially terminal masters, 1 of the 10 does an academic masters and proceeds to PhD, of the other 9 some may come back for an applied degree.

A Taxonomy for Computer Science

There now exists a body of knowledge which may be designated 'Computer Science', but people generally disagree as to what computer science may be. In the following table a classification is proposed which covers the subject but hardware is not included since that is considered to be the province of the electrical engineer. This point of view is of course open to objection but we do need to start somewhere and it should not be difficult to find an equivalent hardware oriented formulation.

The classification is summarized in table 1.

The mathematics for computing is summarized in appendix 1, and is to be discussed and justified in a separate paper.

It must be noted that the applications areas each have an associated body of knowledge which must be acquired by the student. Such knowledge is typically obtained by ancillary courses or a second major at the bachelor's level. In particular

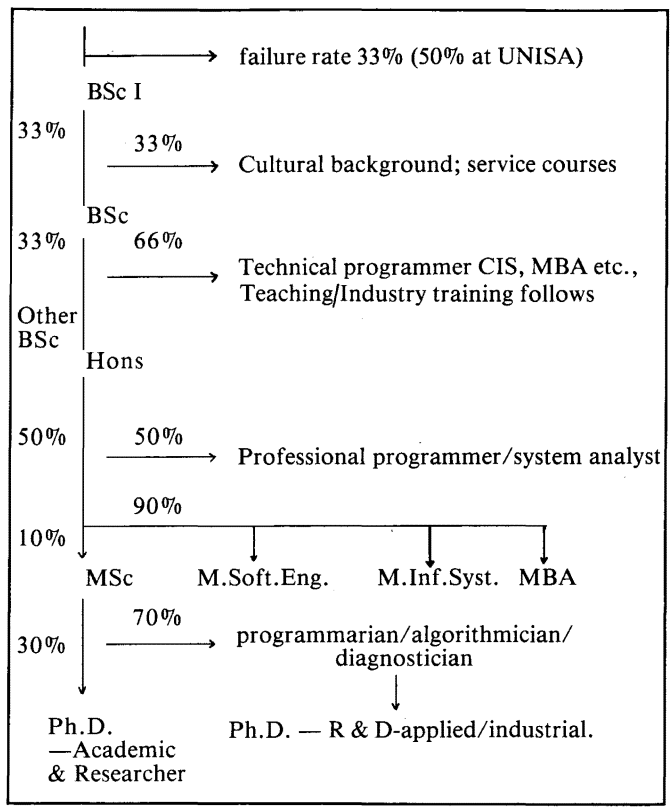

FIGURE 1: Input/Output diagram for complete degrees structure

the need for statistics in artificial intelligence, industrial economy/psychology/law in data/information processing, and mathematical analysis in graphics and numerical analysis cannot be over emphasized.

Core Curricula for Computer Science

The core curricula for computer science and mathematics are summarized per year in table 2. The requirements of other subjects to complete a year's curriculum are not considered but the remarks in the section above give an indication of what is required. The computer science syllabus are considered in the next section, and a brief look at the master's degree concludes this section.

Many well-known topics are absent from the core curricula, in particular compiler design and operating systems. It is my contention, further discussed in the next section, that these are specialist topics. If we follow Wegner's lead in considering the master's degree to be just that, and not a PhD-failed, then these topics cannot be properly covered at a pre-masters level, and the work is included at that level at the expense of core material so that a lot of time is wasted on material which should in any case not occur in options but be mastered by all the students. A typical masters then is devoted to one such subject, to master the subject itself and in a practical or theoretical (semi) dissertation to demonstrate acquaintance with methods of research.

Syllabi for Computer Science

The following are basic assumptions of the syllabi:

1. The first year must be a general introduction to the whole area of computing, and is not considered in detail.
2. The material must be presented from the concrete to the abstract: an early introduction of abstraction yields parroting not knowledge.
3. Iteration over material is required by the learning process.
4. The areas denoted TC & PC must be offered in an integrated fashion.
5. Program languages are covered in the lectures, programming languages in the practicals.
6. Three-hour practical sessions are formally required: two at the second year level, three at the third year level and four at the honours level. Aim: Transference of know-how.
7. The AC-options cover only core material, other applications are to be added as options depending on departmental interests and availability of lecturers.

The second year concentrates on data items, hence an assembler language is studied, and the combination of items in linear structures — vectors and files and hence Pascal is studied.

Subroutines and functions are covered in a typical von Neumann environment. This of course goes well with the iterative approach for linear structures and the correctness by loop invariants are covered as formal proofs, a lá mathematics, in the lectures. The student is presented with an algorithm and its proof of correctness, justification for termination and indication at least of complexity. The student is not to reinvent the wheel again and again, but must use the material in the practicals to solve problems.

Predicate transformations and a top-down presentation of the formal material helps to inculcate good habits, of which proper documentation is the most important. But a top-down development of a solution can only work if you have a very shrewd idea of where you are going, else you are doing research. This is just too much to expect from the undergraduate.

The language(s) used in the practical work is not important, and Cobol might well be one of them, or Ada for that matter.

Compared to standard syllabuses the proposed syllabus does not attempt to cover all data structures, but greater attention is paid to algorithms.

The third year concentrates on trees and hence recursion (via Lisp), and the student is introduced to non-procedural programming via Prolog. Thus an approach is made to non-von Neumann architecture and the Lisp itself is useful as an introduction not only to functional programming — it is assumed that the Prog-feature will not be used — but also to operational semantics.

MC.	Mathematics (Metatheory) for computing: (q.v.) *Techniques of proofs* Logic; automata, grammars & languages; algebra; linear spaces; recursive function theory; lambda-calculus etc.
TC.	Theory of computing: *proofs about programs* Semantics: operational, denotational, axiomatic; syntax; proofs; equivalence; schemata etc. Models: automata; architectures.
PC.	Pure Computing: *programs* Data structures/types; algorithms-efficiency; program(ming) languages; architecture; control structures; programming methodology, etc.
AC.	Applications: *using programs to solve problems* Data processing; information processing; artificial intelligence; numerical analysis; graphics; hardware design; ...

TABLE 1: Taxonomy

BSc II: Sequential Data Structures, Iterative Control	
MC:	Vectors and matrices; number theory; combinatorics; finite automata (FA) & regular expressions (re), Turing machines (Tm) & halt problem; model theoretic propositional & predicate logic.
TC:	Loop invariants, predicate transforms; proofs of effectiveness & efficiency.
PC:	Assembler language programming: data items — strings of bits, characters, numbers — loops & subroutines, von Neumann architecture. Pascal: vectors, records, files; functions.
AC:	Applied number crunching: percentages, statistics, annuities etc. File processing: sorting, merging updating etc. I/O layout.
BSc III: Sets & Trees, Recursive & Backtrack Control	
MC:	Axiomatic propositional & predicate logic; sets, relations, (partial) order, etc; algebra: (semi-) groups, rings, fields, ideals; push-down automata (PDA), types 0-3 grammars & languages.
TC:	Synax (-analysis); effectiveness, efficiency, equivalence by various recursive proof techniques; semantics: operational, axiomatic, schemata.
PC:	Sets, trees, data types; Prolog & non-procedural programming; (pure) Lisp & recursive programming; non-von Neumann architecture-I; and-or trees and backtracking.
AC:	Sets & random files; symbol & program manipulation — editors; hierarchical & relational databases.
BSc Hons: Graphs, Nondeterminism & Concurrency	
MC:	Lattices; recursive function theory, lambda-calculus and Turing machines; graph theory; introduction to topology.
TC:	Denotational semantics; schemata, Petri-nets, etc; efficiency-effectiveness-equivalence: pragmatics.
PC:	Functional programming, lazy functional programming & coroutines; non-procedural & applicative programming; non-von Neumann II, concurrency, etc.
AC:	Networks; database-network model; distributed processing; VLSI (Very Large Scale Integration); AI (Artificial Intelligence); decision procedures, expert systems; PC(Hons) methods applied to structures of BSc II & III.

TABLE II: Core Curricula

The third year differs from the standard syllabus in that comparative programming languages are not covered, neither are operating systems except in so far as they may be covered in the practical sessions — i.e. emphasis on what they do rather than how. Programming languages are to be covered at the honours level when the student has been taught and have used at least four different languages. Data base concepts are, for similar reasons, also covered in the practical part of the syllabus.

Also translators are not covered specifically but with the specified study of automata and trees and backtracking it is obvious that syntax analysis will be covered as an application. This also gives a good starting point for comparative complexity studies and, where possible, proofs of equivalence.

The honours course introduces material on graphs and denotational semantics, is a revision of the undergraduate work, and has a distinctive number of compulsory applications of which artificial intelligence is the most important. The specific AI topics to be covered include theorem proving/problem solving, program generation — linking with functional programming, and expert systems. Topics in cognition and pattern matching should be left for the masters.

Software Engineering/Systems Analysis and Design: There are two approaches to this area — the soft and hard. The soft option should not be available to a computer science department but may be left to the engineers and the management scientists. The hard option presupposes a proper grounding in the fundamentals of computer science and can only be taken at the masters level.

Conclusion: The syllabuses outlined constitute a 90 degree change in course — e.g. data structures, or languages, are not studied in one year but over a number of years. The question of appropriate texts then arise. The solution is to use a book over more than one year, or maybe publishers could publish their books by the chapter so that the lecturer can set up his own text by selecting appropriate material. (This is an idea for texts in machine readable form).

Mathematics for Computing

Concepts: The function as — graph
process
object.

Approaches:
Formal abstract = axiomatic
Numeric
Formal manipulation i.e. calculi
Applied = modelling
Constructive & Finitary.

Topics

— Propositional & predicate logic — proof & model theoretic
— Formal automata, grammars & languages
— Recursive function theory — fix points
— Linear spaces: vectors, matrices, determinants
— Decidability etc.: Turing machines & lambda-calculus
— Naive set, graph, number theory; combinatorics
— Algebra: (semi-)groups, rings, fields, ideals; lattices etc.

Optional — but surely expected:
— Differential & integral calculus
— Point set topology — metric spaces
— Universal algebras
— Mathematical statistics & probability
— Optimization; linear, non-linear, dynamic programming.

APPENDIX 1

Bibliography

'ALVEY REPORT': A Programme for Advanced Information Technology, The Report of The Alvey Committee, Dept. of Industry, HM Stationary Office, London, 1982.

TRAUB, JF (Ed): Quo vadimus: computer science in a decade, comm ACM, *24*, 6, June 1981, pp 351 — 369 (The 'Quo Vadimus Report').

DENNING, PJ (Ed): A discipline in crisis, Comm ACM, *24*, 6, June 1981, pp 370 — 374 (The 'Snowbird Report').

BENNET, JM: What is computer science? 2nd Australian Computer Science Conf. Hobart, 1979.

RALSTON, A; Shaw, M: Curriculum '79 — is computer science really that unmathematical?

UNIVERSITY GRANTS COMMITTEE — Mathematical Science Sub-Committee: Report on Computer Science, May 1982, London.

THE FIFTH GENERATION COMPUTER PROJECT — special section in Communications of the ACM, *26*, 9, Sept. 1983, pp 629 — 645.

POSTMA, SW; Laing IDG: Programmers, programmarians and programmaticasters — on the training of — Procs, WCCE, Lausanne 1981, also in AEDS Monitor *20*, 4 — 6, 1981, pp 28 — 31.

New reading

Mathematical Foundations of Programming **by Frank S Beckman. Published by Addison-Wesley, Reading, Massachusetts, 1980. 443 pages, exercises, chapter references, index.**

THE phrase 'mathematical maturity' is used to describe a way of thinking that is required not only of computer scientists but also of programmers, not only of systems analysts, but also of managers of programming projects and thus also of dp managers. Very few people would like to attempt a definition of this desirable quality, but one may indicate a relatively painless way of acquiring it: Peruse the book under review.

In its breadth of coverage (with one omission noted below), in the topics dealt with, in the way all topics are related to actual or potential computing practice, the book contributes to the acquisition of the vitally important quality of mathematical maturity and also a quality that can only be termed algorithmic maturity. In other words, this book contributes to the appreciation of what can but also of what cannot be computed, and the various ways in which a computation may be characterised, realised, criticised, and analysed.

The author covers the area variously known as 'theory of computation' or 'meta-theory of computing', in a descriptive, almost intuitive, but mathematically sound fashion. The topics covered include: The concept 'effective'; functions and sets; recursive functions; computability and its limitations; automata and languages; and computational complexity.

In every chapter the author introduces the main topics by examples (so necessary for the development of the 'feel' for a subject) and then progresses through mathematical descriptions to applications. In the process many sidelines are touched upon which in textbooks are mentioned but then tantalisingly ignored. This is thus a book not only to be read but to be dipped into.

The practitioner in computing will find the book worth reading, not only to acquire the aforementioned maturity, but also to gain a background knowledge of the many developments in progress all over the world which go by the name 'fifth generation'. It is rather unfortunate, however, that the predicate logic receives but scant attention in the book since the Japanese fifth-generation R and D is based on Prolog-programming in logic. The research thrust in America, by the MCC, is to be based on a Lisp foundation and various aspects of this functional style are covered in the book.

For students taking a formal course at honours level in one or more topics covered by the author, the book should be recommended reading in order to put their field of study in perspective. For honours students not taking such topics, the book should be required reading.

STEF W POSTMA

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:

Department of Computer Science
University of South Africa
P.O. Box 392
Pretoria 0001
South Africa

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 a summary of 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 l, 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. BÖHM, 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 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 against 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.

Fifty reprints of each article will be supplied free of charge. Additional copies may be purchased on a reprint order form which will accompany the proofs.

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.

Quaestiones Informaticae

Contents/Inhoud

*Presented at the Third South African Computer Symposium held on 14th to 16th September, 1983.