QI QUAESTIONES INFORMATICAE

Volume 4 Number 2 May 1986

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

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-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

Circulation and Production

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 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.

NEW CS SYLLABUSES AT NATAL (PMB)

Stef W Postma
University of Natal
Pietermaritzburg 3200

The Department of Computer Science of the Pietermaritzburg Campus of the University of Natal is one of the youngest departments in the country, although service courses to the Faculty of Commerce have been available for some years. In 1985 the department is teaching its second lot of third year students, and is preparing to offer an Honours course in 1986. Masters and doctoral students are catered for on an individual basis.

As a result of the experiences gained over the last four years, a slight expansion in the academic staff complement, the fifth generation ferment, and the requirements of the Honours degree, it was felt that the syllabuses [UNIV-85] should be revised. This paper sets out the final results of the revision and attempts to justify some of the decisions reached. It should however be realised that the revision did not involve a change of course amidstreams, but, in many instances, only minor adjustments.

1. GENERAL AIMS AND PRINCIPLES

The general principles for setting up syllabuses as outlined in Postma [Post-84] were followed. The soundness of this approach can only be appreciated if the remarks of Hoare (in Broy and Schmidt [BrSc-82]) are taken to heart (vide also Wulf, Shaw, Hilfinger and Flon [WuSh-81]: software engineering must be based on a solid scientific basis, or else, to paraphrase Dijkstra [Dijk-80] it is merely soft and not engineering). In the three year B.Sc., or four year Honours degree, there are actually few opportunities to do more than lay the foundations for a proper discipline of programming or the practice of informatics.

Computer Science, or Informatics, differs from all other sciences [Hart-81]. Two particular differences and a general aim that have to be appreciated are crucial and are indicated below:

- Experiential component : Computer Science is not an experimental science (cf. Physics) but the student should learn how to conduct experiments - e.g., to establish what this compiler is capable of doing on that computer. Nor is Computer Science an observational science, but this is very important in systems analysis. Computer Science is an experiential science in that e.g., writing a large program can only be experienced, or as Metzger [Metz-73] says - if you have not done it before you are not developing but researching.
  The student then must practice, and experience, different types of program development styles and master the idioms and their combinations cf. Abelson and Sussman [AbSu-84]. This aspect is highlighted in the article of Feldman and Sutherland [FeSu-79].

- Feasibility : Computer Science has two natural boundaries; it is bound by the practicable on the one hand - a movable and moving boundary, and the limits of computability and constructivist mathematics on the other hand - an immovable boundary. The computer scientist/informaticist/informatician must know the limits of his endeavours, and must concentrate on the feasible [Hart-81], i.e. the spectrum from practicable through 'feasible' to constructive; the conceivably executable - definable - procedurally describable on a real or putative computing device.

- Maturity : The programmer, the software engineer and of course the computer scientist requires mathematical maturity ([LiMi-79]; Hoare, Dijkstra, Bauer, etc. - too many to cite), and he requires algorithmic maturity. This maturity is only obtained by training,training from the concrete to the abstract. the latter principle cannot be emphasised enough - see e.g. Beth, quoted in Postma and Laing [PoLa-82]. Furthermore, maturity is only obtained by experience and thus an abstract approach must at some stage be reached; where the coder produces programs the computer scientist should be able to consider the program itself as an object of study.

2. THEMES OF THE COURSE

The syllabuses of every year prescribe what material is to be covered in that year, and the horizontal integration of the topics in a given year is discussed in the next section. The vertical integration of the topics is described in terms of three themes which run from first year to honours.

- **Programming :** idioms - algorithms -> construction : The professional should be able to describe his program, an interrelated, idiomatically combined, selection of algorithms - in the most appropriate language, then implement it effectively. The programming theme then comprises:

  B Sc I: Iterative programming - structured programming
  II: Recursive programming, assemblers
  III: Concurrent and stream programming, operating systems
  Hons: Applicative programming, software engineering

- **Data structures and analysis of algorithms :** The professional programmer should be able to choose appropriate structures and algorithms, on criteria of efficiency and correctness. The data and analysis theme covers, i.a. :

  B Sc I: Using linear data structures, files
  II: Analysis of algorithms on linear data structures, correctness by inductive assertions, implementation: assembler programming. Using tree data structures.
  III: Analysis of recursive programs, and data structures. Axiomatic methods : data types. Unbounded (potentially infinite) data structures. Large systems - data bases.

- **Tools for abstraction :** The basic techniques taught in mathematics need to be augmented (see Beckman [Beck-80], Tremblay and Manohar [TrMa-75], Broy and Schmidt [BrSc-82], Bauer and Wössner [BaWo-82]). Since not only a knowledge of discrete mathematics but rather mathematical/algorithmic maturity is the goal, the formal theme comprises:

  B Sc I: Set Theory - subsets of a universal set. Finite automata, regular grammars and regular sets. Model theoretic propositional calculus : Venn diagrams.
  II: Predicate calculus, axiomatic propositional calculus. Inductive assertion method. Push-down automata, context free grammars and languages.
  III: Lambda-calculus, operational semantics (e.g. SECD). Recursive and structural induction. Introduction to temporal logic.
  Hons: Recursive function theory, fixpoints. Justification of inductive assertion, recursive induction, etc. Correctness of concurrent programs. Formal abstract data types.

3. THE SYLLABUSES

The following is given for each year : The objective of the year's study, the actual syllabuses, notes on material to be covered, approaches, etc.

- **B Sc I :**

Objective : A thorough grounding in iterative programming techniques based on Pascal.
Syllabus : 100 lectures + 26 practicals.

Introduction to computers and computing; algorithms; programming in Pascal; sets; programming involving files and data structures. Number systems and representation of data in the computer; programming techniques and structured programming. Propositional

logic; finite automata; regular expressions.

Approach : The student is to concentrate on applied computing, i.e. using computers to solve problems. This approach trains the student in the basic techniques of using the computer. It is thus suitable for the student taking only one course in computing and forms a basis for the concepts of the subject at more advanced levels.

File processing, e.g. File updating, e.g. adding compound interest to records, etc. should be covered as well as an introduction to random files.

Automata theory should not only be seen as a theory but as a source of interesting programming applications - vide Mallozzi and DeLillo [MaDe-84].

- **B Sc II - Prerequisites : Computer Science 1; Mathematics 1.**

Objective : A thorough grounding in recursive programming techniques based on a functional subset of Lisp. An introduction to the implementation (assembler) and analysis (complexity, correctness) of algorithms on data structures.

Syllabus : 100 lectures + 26 tutorials + 26 practicals.

Design and analysis of algorithms with application to data structures; recursive programming and symbol manipulation; assembler programming and pointers - systems software; predicate logic - correctness of programs; applied numerical methods; error propagation; complexity; correctness. Push-down automata; regular and context-free grammars - syntax analysis.

Approach : The second year student studies, inter alia, the algorithms used in the first year w.r.t. implementation, time (and space) complexity, and their formal correctness by means of the method of inductive assertions. Most of the proofs have of course to be supplied by the lecturer.

He learns to use the idioms of recursive programming, applied to trees and symbol manipulation. This gives a good grounding for later studies of artificial intelligence. The study of pda's are not only interesting for the light they throw on computability, but ties in with the study of systems software via syntax analysis and an introduction to program translation as an example of program transformation.The latter again ties in with the functionals studied in the recursive programming.

Only a small subset of Lisp, and a rather pure functional subset need to be studied - vide Henderson [Hend-80] chapters 1-3 and Postma [Post-85].

At the end of the second year of study the student should be comfortable with the idea of a program, a procedure, an algorithm as an object that can be studied, manipulated, transformed, passed as an argument and returned as a value.

- **B Sc III - Prerequisite: Computer Science 2.**
  Corequisite : Mathematics 2.

Objective : A thorough grounding in concurrent, parallel and stream programming techniques, i.e. basics of real-time programming. Analysis of algorithms. An introduction to program(ming) language semantics.

Syllabus : 150 lectures + 26 tutorials + 26 practicals.

Operating systems; networks and data communications; computer architectures. Concurrency; real;-time and stream programming; architectures. Systems analysis and design; design, implementation and theory of data bases; analysis and correctness of algorithms and data structures. Theory of computing applied to the design and analysis of

algorithms, and programming languages; lambda-calculus and operational semantics; temporal logic.

Approach : At the end of the third year the student has covered the basic types of programming - iteration, recursive, concurrent, and should appreciate some of the problems of large scale programming. The latter is obtained by setting a group programming project in the third year, and careful attention to program construction methods during the practical sessions.

Analysis of programs is extended to analysis of the means of expressing programs, i.e., program language semantics. The approach is again from the concrete to the abstract, thus denotational semantics is only encountered at the Honours level.

The temporal logic is only briefly to be touched on, and is almost purely to be an introduction to the Honours, thus should not be emphasized in the examination. The lambda calculus is to include the Church-Rosser Theorem as a fact - the proof is not to be included in the syllabus. An extention to combinatory logic and reduction architectures cannot be covered at the third year level.

- **B Sc Honours - Prerequisites: Computer Science 3, some mathematical maturity** *- i.e. to the satisfaction of the Head of Department.*

Objective : Twinfold - solid grounding for the software engineer/programmarian/diagnostician, i.e. the professional; and solid basis for the M Sc which should be an 'experiential' degree, *not* a Ph. D. (failed) [Wegn-72]. The Honours degree is the degree that is variously recognised to be the minimum professional qualification.

Syllabus : 250 lectures + practicals + seminars.

- *A course in pure or applied mathematics* or *formal logic and recursive function theory.*
- *A selection of four courses from :*
  1. *Efficiency, effectiveness and equivalence of programs.*
  2. *Program languages and architectures.*
  3. *Axiomatic, operational and denotational semantics.*
  4. *Artificial intelligence.*
  5. *Data structures and data base theory.*
  6. *Software engineering and program construction.*
  7. *An honours course offered at the Durban campus, or any special topic offered.*
  8. *Any other course - on approval by the Head of Department.*
- *A practical programming project, fully documented, in which a subminimum mark must be attained.*

Approach : The actual topics available in a given year will depend on the availability of staff and visitors, and students' interests.

On completion of the Honours degree the student should have acquired the necessary background information to either :

1. becoming a software engineer, capable of applying known scientific principles to the development of large software systems, or
2. continuing a career in pure or applied research, where in applied research he will be concerned with the use of computers on solving problems, and thus may well find his way to a big industrial or computing company or a Technikon; and in pure research he will be concerned with any and all aspects of feasible algorithms and information structures as objects, using inductive reasoning, and deductive reasoning but above all reductive reasoning to establish the feasible.

Given the current, and the future, acute shortage of trained people [Vand-85] the student

should be trained to be as effective as possible.

4. NOTES ON THE TOPICS

a) Formal logic and recursive function theory -

- Initially the logic will mainly cover propositional and predicate logic with some temporal logic. As the undergraduate logic programme develops the emphasis should change to more and more temporal logic and applications.

- The recursive function theory yields additional insights into computability and justification of Church's thesis.

- The fixpoint theorem is not only one of the central theorems in theoretical computer science, but affects program language design and is used in the justification of various formal techniques of program verification. The relation of this theory to denotational semantics has been studied by Scott [Scot-82].

b1) Efficiency, effectiveness and equivalence of programs : The first two topics - efficiency and effectiveness - have been encountered at the undergraduate level. Depending on the lecturer's interests, and the class's background the course may be mainly on efficiency (interest related to mathematical analysis) or effectiveness or equivalence (interest related to algebra). A combination of the effective and equivalence with applications to program transformation in a theory of, say, predicate transformers or recursive functions or denotational semantics is probably preferable to other options.

b2) Program languages and architectures : If b3 (semantics) is not taken then this course should concentrate on denotational description of programming languages with applications to standard architectures. If b3 is taken this course will probably concentrate on applicative programming with applications to, e.g. operating systems and reductive architectures based on combinators.

b3) Axiomatic, operational and denotational semantics : The student will have some background in the lambda-calculus and SECD machines, and the course is therefore to cover the axiomatic approach, but is to concentrate on the denotational specification of programming languages. Both procedural and functional languages are to be considered and the overall appraoch to be that of 'complementary definitions' [Dona-76].

b4) Artificial Intelligence : Basic symbol manipulation techniques have been covered at the undergraduate level, and logic at various levels. Students who wish to specialize in AI are furthermore encouraged to include statistics in their curriculum. The course will concentrate on production systems, and then, depending on lecturer and students, on either theorem proving with applications to program correctness or expert systems.

b5) Data structures and database theory : If efficiency is not covered in b1 then this course should contain a fair amount on efficiency. The course however is meant to cover the axiomatic, set-theoretic and algebraic approaches to data in all its forms and manipulations.

b6) Software engineering and program construction : Where most of the other courses are concerned with programming in the small, this course is concerned with programming in the large. If it is not taken then aspects of it must be incorporated in C: programming project, by means of seminars.

b7) Durban course/special topic : Allows the student to take a specialist topic at Durban, e.g. image processing; or a specialist topic offered by a visitor. (Potential visitors should note that Pietermaritzburg offers cheap living, and is 80 kilometres from either sea or mountains.)

b8) **Any other course** : Such a course may be at the post-graduate level, if the student is admitted by that department, but could also be at the undergraduate level, say mathematical statistics 2 for a student who has decided that AI is his 'scene'. Thus either a potential applications area may be studied or ancillaries may be catered for.

C. **Programming project** :
A fully documented programming project is to be completed, including at least one structured walk-thru presentation as a seminar. The documentation must cover :

A - Analysis of the problem and specifications of program.
B - handBook for using the program.
C - Coding.
D - Debugging and Development.
E - Evaluation with suggestions for Extentions, etc.
F - Formal efficiency and correctness.

If the student is so inclined, he may use undergraduate students to help in the coding phase.

5. CRITICISM

The immediate expected response to the syllabus 'it is too theoretic' is refuted by the authors cited, and by pointing out that it would indeed be too theoretical if the practicals were incorrectly approached. In particular much of the teaching of programming languages per se could be done in these periods. It should further be noted that training for research has a high priority at the University of Natal, which is rated [CSIR-85] ahead of many universities, excepting Witwatersrand and Cape Town, as a research institution.

To judge the syllabus one should realize that the material covered has been chosen with the idea that the student should at the end of the course be able to read at least some journals in order to keep up to date (e.g. Communications of the ACM, Journal of British Computer Society), and also have acquired insight into the techniques which may be in vogue ten years after graduation.

A prognostication of important directions with ratings of the syllabus follows :

1. Functional / applicative programming - well covered.
2. Formal approaches - well covered.
3. Program development - well covered.
4. Artificial intelligence / expert systems - solid basis.
5. Data base - systems analysis - solid basis.
6. Architectures - covered.
7. IPSE/Software engineering - basics.
8. Nonprocedural / 4th generation / Prolog - basics.

A comparison with topics taught at the Australian universities given by Postma [Post-84b] indicates that the syllabus, for a small university, compares very favourably in that all the major areas are covered (exceptions in brackets): *Architecture; artificial intelligence; operating systems; programming languages; functional programming; algorithms-complexity; numerical; (compilers)-automata; computability-meta- theory; (information systems) software engineering.*

Finally a comparison with the syllabus requirements of the ACM/IEEE [MuDa-84] draft accreditation report shows that the syllabus presented is better than that available at many universities in the United States of America.

ACKNOWLEDGEMENTS

The syllabus was finalised with the cooperation of the whole department - N C K Phillips, giving input and comment all the way from Illinois where he is on sabbatical, G Finnie, R

Dempster and C Scogings. Without positive and critical input from everybody the syllabus would have mirrored the prejudices of an individual.

REFERENCES

[AbSu-85] Abelson, H; Sussman, G J (with Sussman, J): Structure and Interpretation of Computer Programs; The MIT Press, Cambridge Mass., 1985, ISBN: 0-262-01077-1.

[BaWo-82] Bauer, F L; Wössner, H: Algorithmic Language and Program Development; Springer-Verlag, Berlin, 1982, ISBN: 3-540-11148-4.

[Beck-80] Beckman, F S : Mathematical Foundations of Programming; Addison-Wesley Publ. Co., Reading, Mass., 1980, ISBN: 0-201-14462-X.

[BrSc-82] Broy, M; Schmidt, G (Eds): Theoretical Foundations of Programming Methodology; D. Reidel Publ. Co., Dordrecht, Holland, 1982, ISBN: 90-277-1460-6.

[CSIR-85] CSIR - Foundation for Research Developmant: Report of the Main Research Support Programme; S.A. National Scientific Programmes Report No. 101E, February 1985, ISBN: 0-7988-3372-6

[Dijk-80] Dijkstra, E W: My hopes of computing science; pp 95-110 in [FrLe-80].

[Dona-76] Donahue, J E: Complementary Definitions of Programming Language Semantics; Springer-Verlag, Berlin, 1976, LNCS 42, ISBN: 3-540-07628-X.

[FeSu-79] Feldman, J A; Sutherland, W R (Eds): Rejuvenating experimental computer science; *Comm. ACM* 22, 9, September 1979, pp 497-502.

[FrLe-80] Freeman, H; Lewis, P M (II) (Eds): Software Engineering: Academic Press, New York, NY, 1980, ISBN: 0-12-267160-0.

[Hart-81] Hartmanis, J: Quoted participant in: Traub, J F (Ed): Quo Vadimus: Computer Science in a Decade; *Comm. ACM* 24, 6, June 1981, pp 351-369.

[Hend-80] Henderson, P: Functional Programming Application and Implementation; Prentice-Hall, Englewood Cliffs, NJ, 1980, ISBN: 0-13-331579-7.

[LiMi-79] Linger, R C; Mills, H D; Witt, B I: Structured Programming Theory and Practice; Addison-Wesley Publ. Co., Reading, Mass., 1979, ISBN: 0-201-14461-1.

[MaDe-84] Mallozzi, J S; De Lille, N J: Computability with Pascal; Prentice-Hall Inc., Englewood Cliffs, NJ, 1984, ISBN: 0-13-164443-2.

[Metz-73] Metzger, P W: Managing a Programming Project: Prentice-Hall Inc., Englewood Cliffs, NJ, 1973, ISBN: 0-13-550756-1.

[Post-84] Postma, S W: Syllabi for Computer Science as a scientific discipline; *Quaestiones Informaticae*,3, 2, July 1984, pp 3-5.

[Post-84b] Postma, S W: The study of Computer Sciences in Australia at tertiary education levels; pp 127-136 in: Procs. 14th CSLA Conf. - Hluhluwe - June 1984.

[Post-85] Postma, S W: Introduction to Functional Lisp; Technical Report PMB-TR/85-3, Dept Computer Science University of Natal, 1985, ISN: 0-86980-448-0.

[PoLa-82] Postma, S W; Laing, I D G: Programmers, programmarians and programmaticasters - on the training of - ;Procs., WCCE, Lausanne 1981, also in *AEDS Monitor* 20, 4-6, 1981, pp 28-31.

[Scot-82] Scott, D S: Lectures on mathematical theory of computation; pp 145-292 in [BrSc-82]

[TrMa-75] Tremblay J P; Manohar, R P: Discrete Mathematical Structures with Applications to Computer Science: McGraw-Hill, New York, NY, 1975 ISBN: 0-07-065142-6.

[UNIV-85] University of Natal Calender 1985, pp K60-K61.

[Vand-85] Van der Veer, G: Quoted in Computer Week, 8, 18, 13 May 1985, Frontpage.

[Wegn-72] Wegner, P: A view of Computer Science education; pp 1515-1522 in: Information Processing 71; North Holland Publ. Co, Amsterdam, 1972.

[WuSh-81] Wulf, W A; Shaw, M; Hilfinger, P N; Flon, L: Fundamental Structures of Computer Science; Addison-Wesley Publ. Co., Reading, Mass., 1981, ISBN: 0-201-08725-1.

BOOK REVIEW

The AI Business: Commercial Uses of Artificial Intelligence, Patrick Henry Winston and Karen A. Prendergast (editors), (The MIT Press, Cambridge, Massachusetts, 1984; ISBN 0–262–23117–4); 324 pages, $19.95.

Reviewed by: Philip Machanick, *University of the Witwatersrand, Johannesburg 2001*

The AI Business is essentially the proceedings of a colloquium held at MIT, which brought together academics, financiers, industrial R&D people and end-users. Aimed at a non-technical audience, it says little new about AI (though Alan Kay offers some insights—if mainly into how big business sees research).

That is not to say the book is not entirely without interest: it is a good starting point for reading up an area you are not familiar with. Even so, as a survey it lacks balance. Expert systems are, of course, extensively covered, though with emphasis on selected case-studies (XCON, DIPMETER ADVISOR, CADUCEUS, The Programmer's Apprentice). Natural language receives some coverage, with emphasis on front ends—Roger Schank's Cognitive Systems, Inc. and Artificial Intelligence Corp.'s INTELLECT.

The other major section is robotics. This area has appeared less and less in the mainstream AI literature and more and more in its own specialized journals. From these papers and the discussion, it is obvious why. It is only comparatively recently that AI work in vision (although applied in research by Michie *et al.* in Edinburgh over ten years ago [1]) has found its way into industrial robots. Scant mention is made of planning systems (remember the blocks world?). Most of the robots in use now are not using AI at all, and Paul M. Russo of General Electric takes pains to minimize the impact of AI on his field.

There are some glaring omissions: given that the colloquium was looking at commercialization, the lack of representation of purveyors of AI tools is strange. Even AI workstations are given little mention. Daisy Systems Corp., who make a VLSI design workstation are there, but deny being an AI company. The only other representative of this area is John Seely Brown of Xerox, whose main contribution —besides a lengthy historical review—is an unconvincing explanation of why the Alto was not commercially viable.

Marvin Minsky is very much in the wilderness at this colloquium. Everyone else is either optimistic or pleased with what they were doing as long as no one calls it AI. Minsky's problem is the lack of basic research: "Marvellous things are happening, but many of them are happening because of research done ten or fifteen years ago ... In 1970 memory cost a penny a bit or so ... Processors are much cheaper and faster. It is odd, then, that industry is using software developed a long time ago under other conditions."

If the book is not an altogether convincing survey, it is less convincing as a harbinger of vast profits from AI. Minsky and Kay are not fully answered in their misgivings about business's understanding of basic research, and business remains skeptical of the commercial value of AI, besides obviously successful applications such as DIPMETER ADVISOR and XCON. Still, it is good light reading and will help to take the mystery out of AI for the naïve reader in a slightly less melodramatic fashion than another well-known book [2].

References

1. Michie D., *On Machine Intelligence*, Edinburgh University Press, Edinburgh, 1974. ISBN 0–85224–262–X.

2. Feigenbaum E. A. and McCorduck P., *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*, Addison-Wesley, Reading, Massachusetts, 1983. ISBN 0–201–11519–0.