ISSN 0254-2757

QI QUÆSTIONES INFORMATICÆ

Volume 5 • Number 1 April 1987

G.R. Finnie	On Learning Styles and Novice Computer Use	1
P.S. Kritzinger	Local Area Networks in Perspective	11
S. Berman	Semantic Information Management	19
P.C. Pirow	Research Computeracy	23
C.H. Hoogendoorn	Experience with Teaching Software Engineering	36
C. Levieux	Education Rather than Training	41
D. Podevyn	Decision Tables as a Knowledge Representation Formalism	46
J. Roos	The Protocol Specification Language ESTELLE	51
L.J. van der Vegte	The Development of a Syntax Checker for LOTOS	63
	BOOK REVIEWS	71

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

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.

EDUCATION RATHER THAN TRAINING

Craig Levieux
Department of Computer Science
University of the Witwatersrand

ABSTRACT

The role of teachers in a University Computer Science course is examined. The case for education rather than training is put. It is argued that current methods of teaching are not as effective as they could be. A case study of a different approach is presented and the results evaluated.

1. INTRODUCTION

If Computer Science graduates are to provide a lasting contribution to the computing industry they are going to have to adapt and change with the technology. It is thus vital that they are educated rather than trained. The current situation is examined and found to be wanting. A case study where some improvements were attempted is described together with an analysis of the problems encountered. The conclusion reached is that since it is the students that have to do the learning it is necessary to gain their confidence and co-operation when changes to the methods of teaching are attempted.

2. THE NEED FOR EDUCATION

Computer Science departments are often criticised by industry for producing graduates who 'know nothing' or are not useful. This argument is not new [1] nor is it unique to South Africa. Martin [2] states

> *I spend much of my time with data processing managers, and its a very frequent comment from them that they are not getting what they want from the universities.*

It is also not surprising. Archer [3], in a survey of what business and industry expected from Computer Science graduates found that by far the most popular subjects were Systems Analysis, File Processing, Data Structures, Data Base Processing and Management Information Systems. These were wanted by over 80% of the respondents. In contrast Discrete Mathematical Structures was exepected by only 4,8% of respondents.

There is obviously some confusion as to what Computer Science is and thus probably a need for better communication between the universities and industry, but that need not concern us here. MacGregor [1] points out that that the requirements of commerce for graduates should be met by Data Processing or Information Systems Departments. He then goes on to describe the role that qualified Computer Scientists can expect to play, but restricts his remarks to those with a B.Sc. (Hons) or higher degree.

> *It is a commonly accepted view that a three year degree does not produce a person trained in any specific discipline but having been introduced to some topics at an intermediate level.*

When entering arguments of this sort it is easy to become too involved with what we should or should not be teaching, whilst losing sight of the fact that it is neither the function of, nor should it be the objective of a Computer Science Department to *train* anyone for any specific job. Rather the objective should be to *educate scientists*. Producing graduates with heads full of facts but with no real understanding would be creating a wasting asset – these facts will become irrelevant sooner or later. In a subject that is developing as fast as Computer Science this is likely to happen sooner rather than later. It should not matter that a graduate is not immediately useful in the job situation, because given a good education it should be possible to learn the job in a short time. In addition the graduate should bring new insight and understanding into the job. When advances are made or the technology changes the graduate would be in a position to assess the

Quæstiones Informaticæ, Vol 5 No 1, PP 41-45, ISSN 0254-2757

consequences of the change. A person trained specifically for a job would not have the background to do this. Martin [4] complains that "there is such a vast array of analysts, programmers, and DP managers who are asleep, who are not changing their ways". If we concentrate on training we are in danger of perpetuating this kind of problem.

3. THE CURRENT SITUATION

There is a growing consensus that there is too much emphasis on teaching and not enough emphasis on learning at university, and that scientific cources are light on concepts and overloaded with facts [5, 6, 7, 8]. It would seem that Computer Science is not immune to these problems. Brookshear [9] writes

> *... instead of presenting the knowledge of today in the context of a current (and still imperfect) state in an evolving system, we find ourselves preaching the methodologies of today as rules that must be followed.*

This state of affairs, whilst not desirable, is perfectly understandable. The body of knowledge in the computing area is expanding at a rapid rate. There is no time to teach all there is to know about a topic in the time available, so we rush through teaching as much as we can. This leads to concentration on facts (which are quick to teach) and the avoidance of concepts (which the student takes time to absorb). The net result is that the student learns this large body of facts and regurgitates them in the exams. They are then forgotten two weeks later, which is just as well, because they will probably be out of date within five years.

It is common to find students who are afraid or unable to work things out for themselves. This attitude can persist to the most ludicrous extremes – some students plague the lecturer about every last little and trivial detail of an assignment. They are terrified of making a minor mistake. At the other end of the spectrum there are far too few students who are prepared to explore the unknown.

It is also not uncommon to find that students are unable to transport knowledge from one domain to another. For example, when dealing with user input, what percentage of students make use of their knowledge of lexical analysis and parsing from the compiler course? How many students are able to take what they know about the running time of algorithms (learnt in a data structures course) and their knowledge of disk drives (from a machine organisation course) to deduce that the bottle-neck in a physical database design is likely to lie with disk accesses? Brookshear [9] gives a reason for this

> *We work hard to sectionalise the material into digestible units (and this is good), but in so doing we should realise that we are making it easier for our students to miss relationships among the topics we cover.*

A brief glance at recent examination papers should convince the interested reader that the majority of questions test the students' recall and comprehension – ie the lowest levels of Bloom's Taxonomy [10]. Questions that test application, analysis, synthesis and evaluation are all too often missing.

In order to improve the situation our objective should be to produce graduates who have a thorough grounding in the *principles of the subject* and who are *able to use* these principles to deduce the required results in any specific situation.

The necessity for teaching principles is argued by Shaw [11], Dertouzos [12] and Strachey [13], who writes

> *The premise [is] it is clearly wrong to teach undergraduates the state of the art; one should teach them things which will still be valid in 20 years time: the fundamental concepts and underlying principles. Anything else is dishonest.*

There does not seem to be much evidence of support in the Computer Science community, with the notable exception of Papert [14], for the idea of getting the students to develop their powers of conceptual thought. Other authors [5, 15] have addressed the problem. This should be the most important part of a university Science course. Graduates should have the ability to analyse, synthesise and evaluate. These abilities will stand them in good stead not for one or two years or even twenty years, but rather a whole lifetime. It is far more important to develop

in students a hunger for knowledge and the ability to obtain it for themselves by reading, questioning and thinking than it is to teach them anything at all.

4. AN ATTEMPT AT SOME IMPROVEMENTS

In order to address some of the problems described above a number of changes were made to the topic Programming Languages given to third year undergraduate students at the University of the Witwatersrand in 1985.

- The course was carefully examined to try and determine the fundamental principles of the subject. Based upon this examination the amount of material covered was reduced.

- Small group discussions were introduced. It is now widely accepted that lecturing, whilst it has its place, is often a poor way to transmit knowledge. In the small group discussions it was hoped that the students would learn to work together and to perceive the benefits of doing so especially where it came to problem solving. It was also hoped that the students would get to discover things for themselves.

- The students had to complete three programming assignments during the year, each of which was in a different langauage. In an effort to reduce the compartmentalisation of knowledge the assignments were all related to the Translators topic that the students were covering concurrently. Thus the assignments consisted of writing a simple interpreter, parser and lexical analyser. These were often written before the requisite material had been covered in class. Thus students were allowed to struggle with a problem before a technique for dealing with it was taught. It was hoped that they would then appreciate the usefulness of the technique.

- The assignments were carefully matched to the languages in order to try and bring out their strong and their weak points.

- The students were required to mark each other's assignments, after which they were marked by the lecturer. Marks were awarded for both the assignment and the marking. This idea had first been introduced the previous year with encouraging results (Figure 1). In order to do the marking students were forced to *read* programs and thus accrued the benefits thereof [16]. In addition the students gained a clearer understanding of what was expected of them in the assignments and also how to distinguish between good and bad programs.

- In previous years the students had been required to submit a single long comparative essay. It had been found that the topic was too general and thus the amount of material to be covered was too large. The result was that the essays were often a regurgitation of the textbook and contained no original thought.The students were now required to hand in three short essays (each one linked to the assignment) that required extra reading and original thought. An additional benefit of this was that they received feedback and were thus able to improve their essay writing skills through the year.

- The exam questions required more thinking and less writing than those of previous years. To the uninformed reader the exam would have appeared easier than the previous year's, but since the subject material had not been covered directly in class, it was, if anything, more difficult.

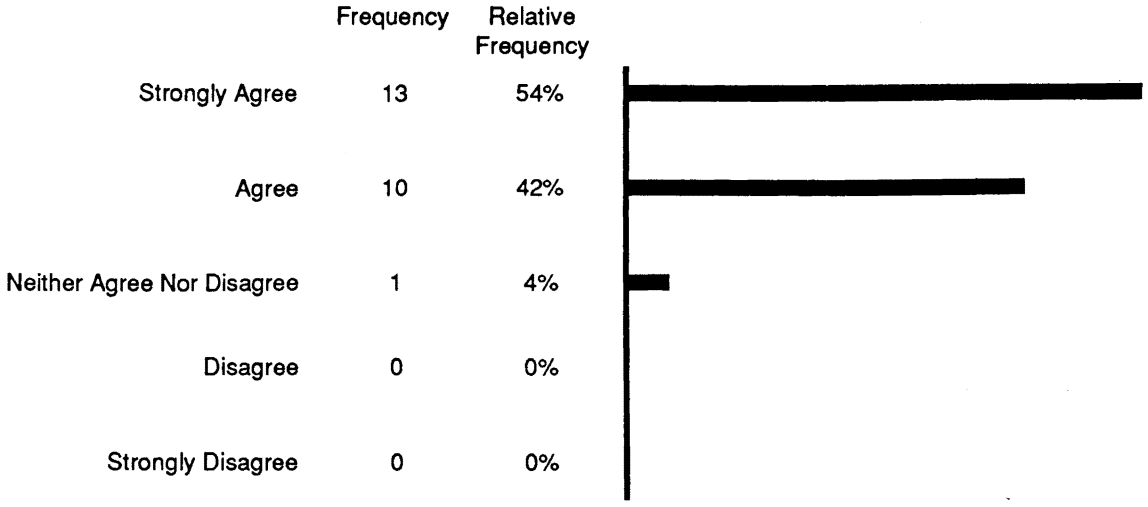

figure 1

Student assesment of lecturer performance. Responses of 24 students to the statement "Marking other students' assignments is a valuable aid to learning".

5. THE RESULTS

Great care was taken to explain to the students that a different approach was being taken and why they should benefit from it in the long run. A small minority of students thrived on the approach taken. Comments such as "The best course I have done" and "The only really worthwhile course I have ever done at university" were heard from this group. The majority, in contrast, were very unhappy with the course. The Head of Department received numerous complaints. The lecturer received lower scores on the voluntary student assesment than he had the previous year, even though he was giving the course for the second time and personally felt that he had done a much better job.

The experiment gave dissapointing results, as have many before it [7], because the abilities and attitudes of the students were not taken into account.

It is obvious that before one starts a course one should discover how much the students know and what their capabilities are. It was assumed that the class would be similar to the previous year's. This was not the case. Two years previously the Maths Department had failed a large number of second year students. Since Maths II is a pre-requisite for Computer Science III this resulted in the 1984 class being some 25% smaller than expected. As the missing students were the marginal ones, the 1984 class was considerably better than average. In contrast, the 1985 class was nearly 100% bigger and almost half the class was made up of students who were weaker than all but one or two from the previous year. Failing to take the abilities of the class into account meant that a significant proportion of the class was unable to cope with the work and thus became negative about the course.

Adams [17] writes that the majority of students want two things from university

- the illusion of learning – the students wants to feel that he learnt a great deal at university but wants this learning to take place with minimum effort on his part.
- a competitive economic advantage – this is perceived as being the degree certificate (which will allow the graduate to obtain a more highly-paid job than the non-graduate) and not the learning that went into obtaining the degree.

These wants manifest themselves in a number of ways

- students are comfortable with the rote learning and regurgitation model. In a course they are introduced to a whole lot of new terminology and facts which they learn and reproduce in an exam. They perceive that they have learnt a lot and the effort has not been large.
- students are preoccupied with marks. Not only passing, but the awards of bursaries

and prizes and also possibly plum jobs depend on them. Marks become important for their own sake rather than being a reflection of how well the student understands the subject.

By the time a student reaches third year, university life has fallen into a fixed pattern and the student fully expects to complete the degree at the end of the year – the rules of the game are well known and understood. By introducing too many new ideas into the course we were breaking the rules. This upset and alienated the students and undermined the success of the course.

6. CONCLUSION

There are ways of producing better-quality graduates than we are doing at the moment. New models of teaching and learning have shown promise but require the co-operation of the students if they are to succeed. Student attitudes are formed in many ways – in the way they were taught at school, the matric examination system and by society itself. This does not mean to say that these attitudes cannot be changed, but third year is the wrong place to try. It may not be a bad idea to throw the student slightly off-balance but it should not be taken to extremes. It is not our aim to create psychological wrecks. It would be far better to introduce new ideas to first year students. They are in any case expecting university to be different from school and would be more receptive to new ideas. This should also incorporate an overhaul of the curriculum [9].

ACKNOWLEDGEMENTS

I would like to thank Conrad Mueller for his contribution to the course. Philip Machanick and Kees Hoogendoorn read the paper and provided helpful suggestions. Richard Rodseth, Paul Dadswell, Philip Green and Sally Benn pointed out numerous errors.

REFERENCES

1. MacGregor, K.J., (1985), Quo Vadis, Computer Science?, *Quaetiones Informaticae*, **3(3)**,pp 2-8.
2. Martin, J., (1985), DP and Academia: The Communication Gap, Panel Discussion, *CACM*, **28(3)**, pp 256-262.
3. Archer, C.B., (1983), What does Business and Industry Expect from Computer Science Graduates Today, in Proceedings of Fourteenth SIGCSE Technical Symposium on Computer Science Education, *SIGCSE Bulletin*, **15(1)**, pp 82-84.
4. Martin, J., (1985), An Information Systems Manifesto, *CACM*, **28(3)**, pp 252-255.
5. Beard, R.M., (1972), *Teaching and Learning in Higher Education*, Penguin (2nd ed.), Harmondsworth.
6. Fletcher, B.A., (1968), The Aims of University Teaching, in *University Teaching in Transition*, Layton, D. (ed.), Oliver and Boyd, Edinburgh.
7. Gerrans, G.C., (1985), *Teaching Within the Faculty of Science, Memorandum for Discussion and Feedback*, Faculty of Science, University of the Witwatersrand.
8. Gerrans, G.C. et al, (1986), *First Report of the Committee on Teaching*, Faculty of Science, University of the Witwatersrand.
9. Brookshear, J.G., (1985), The University Computer Science Curriculum: Education Versus Training, in Proceedings of Sixteenth SIGCSE Technical Symposium on Computer Science Education, *SIGCSE Bulletin*, **17(1)**, pp 23-30.
10. Bloom, B.S. et al, (1954), *Taxonomy of Educational Objectives. Handbook 1: Cognitive Domain*, McKay, New York.
11. Shaw, M., (1984), Goals for Computer Science Education in the 1980's, in Proceedings of Fifteenth SIGCSE Technical Symposium on Computer Science Education, *SIGCSE Bulletin*, **16(1)**, pp 1.
12. Dertouzos, M., (1985), DP and Academia: The Communication Gap, Panel Discussion, *CACM*, **28(3)**, pp 256-262.
13. Strachey, C., (1969), *Software Engineering Techniques*, Buxton, J.N. and Randell, B. (eds.), NATO Science Committee, Rome, pp 65.
14. Papert, S., (1980), *Mindstorms – Children, Computers and Powerful Ideas*, The Harvester Press, Brighton.
15. Novak, J. and Gowin, D., (1984), *Learning How to Learn*, Cambridge University Press.
16. Weinberg, G.M., (1971), *The Psychology of Computer Programming*, Van Nostrand Reinhold, New York.
17. Adams, W.A., (1980), *The Experience of Teaching and Learning – A Phenomenology of Education*, Psychological Press, Seattle.

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.