21

Informatics in curricula for noninformatics students: engineering and science

Roland Vollmar
Lehrstuhl Informatik für Ingenieure und Naturwissenschaftler
Universität Karlsruhe, Am Fasanengarten 5, D-76128 Karlsruhe
Germany, e-mail: vollmar@ira.uka.de

Jozef Gruska
Faculty of Informatics, Masaryk University, Botanicka 68a 60020 Brno, Czech Republic, e-mail: gruska@informatics.muni.cz

Abstract

In this paper three key issues are discussed concerning the role of informatics in science and engineering curricula. The first issue is that informatics is a fundamentally new methodology of key importance for the future. The second is the dramatic increase in basic information technology skills of freshman which makes treatment of more advanced informatics subjects possible and requires a change in teaching skills. The third key issue is that informatics education should concentrate on fundamental ideas thereby supporting life-long learning.

Keywords

Informatics, natural sciences and engineering, university education, continuing education, noninformatics majors, curriculum (general), curriculum (core), role of CIT

Informatics in Higher Education F. Mulder & T. van Weerts (Eds.)

1 INTRODUCTION

The ever increasing role which informatics plays in society, does not need discussion. Some catchwords already are sufficient to get the right association: Internet, multimedia, electronic banking, electronic commerce, electronic warfare, etc. These, however, only touch the surface of the enormous progress we are witnessing. When discussing educational issues, informatics must be recognized as the new fundamental methodology for science and engineering which can help solve many key problems of science, engineering and mankind. The insights, methodsand tools of informatics are of such great importance in science and engineering that informatics should be seen as giving rise to a third fundamental scientific methodology, taking its place besides the experimental and the theoretical methodologies which have been very successful for centuries. For a more detailed discussion of informatics as a methodology see another paper by Gruska and Vollmar (1997). In each of the scientific and engineering disciplines concepts, methods and topics have emerged which can only be handled with methodologies that informatics, including computing and communication technologies, can provide.

It is obvious that this development must have repercussions for the corresponding curricula. Up to now students of science and engineering have only been taught the very basic skills for working with very basic computing technology. This is far from sufficient. Students and graduates should be able to make use of informatics as a methodology during their studies and afterwards. Fortunately, due to the rapid spreading of Internet communication and information services, one may assume that in the immediate future high-school graduates will have the basic knowledge to work with Internet and to use basic word processing, graphical and database facilities of personal computers. This will allow university education to concentrate on the development of skills in using more advanced technology and, especially, to focus on fundamental issues.

The increase in experience of the coming generation of freshmen will also allow us to concentrate on issues of key importance for life-long process of (re)education. One of the current main obstacles will slowly disappear, or at least diminish significantly: insufficient practical experience and motivation of students which make it impossible to really appreciate and grasp more fundamental concepts, methods and results.

This paper puts forward some thoughts and observations concerning basic engineering and science studies at universities. Its main ideas can also be applied on such newly emerging subjects as 'computational sciences'. We will not discuss in depth the specifics of various combinations of (sometimes very different) subject areas. This would force us to differentiate between a plethora of studies. In the following we will first describe the background and the framework into which the courses of informatics have to be embedded. The topics we consider as fundamental will then be discussed in more detail. And to finish we will present an

estimation of the necessary number of hours needed for special informatics subjects.

2 ENVIRONMENT

The following thoughts and comments are of a quite general nature. This is because we are not competent to discuss either the different cultures of the disciplines or the various levels of higher education in various countries. Therefore we restrict ourselves to university curricula in general.

We will try to be realistic, and not assume that it is easy to change current curricula overnight and add many hours for special courses on informatics. In all fields the development is very fast and therefore there is a continuous struggle for more hours in the curriculum. In addition, at least in some countries, universities are under a permanent pressure from government and industry to shorten the overall time students spend in education. We are aware that in this setting informatics has to compete.

Let us make another general remark about university education. It is not possible and also not meaningful to acquire all (or even most of) the knowledge needed to act in a particular profession during university education. Rather the basics and the methodology should be mastered. Professional knowledge has for the most part to be learned during the professional career. Universities have to lay a fundament which makes continuing education possible. This conviction is at the basis of the following and only minimum requirements for change will be advocated.

3 PROPAEDEUTIC

What can we expect in the next few years of the knowledge and skills in information and communication technology which all students will have at the start, and in the worst case at the end, of their first year of study? Just as students are accustomed to utilize telephones and copiers, they will soon have the know-how for using the infrastructure of data processing. They will know how to operate a personal computer (PC) or a workstation. How to use text processing, basic graphical, spreadsheet and presentation software. How to use e-mail as a basic communication tool and how to use the Internet as a basic information searching tool. Those few beginners who will still lack this know-how, will be able to get it at a central university institution, such as the university computing centre, where the newest information about hardware and software systems will be available.

During their whole study students should follow a variety of extra courses to test and improve their skills in dealing with more advanced technology. As a side-effect, such courses may also very well support an introduction into scientific work as such. As Krüger(1987) mentions, education about data and information bases is

less concerned with questions of the computer use, but more with problems of searching, ordering, structuring and combining large amounts of information, i.e. with methods of modern scientific work.

The mastering of these skills in using computing and communication technology should be obligatory for all university students. However, programming is not necessarily among these skills. Yet , the situation may be much different from one discipline to another. In at least some disciplines - for example in electrical engineering, physics, chemistry and biology - students should be able to write more or less sizeable programs, while in other areas - for example civil engineering - students should just use finite element method tools.

With respect to the teaching of programming we are faced with a dilemma which is dependent on the view we take. On the one hand, one can assume that at least some fundamental know-how of programming is already present. In that case a basic course on programming could be combined with, for example, a course on calculus. On the other hand such a basic programming course could be offered as part of an introduction into informatics. Because at least some simple programming skills should be already mastered in the first semester, we recommend the first approach mentioned in combination with laboratory work in programming, especially for those students who have to learn programming from scratch.

4 THE CORE

Is it at all possible to detail the contents of courses, if there is such a rapid development of informatics as is claimed by various persons?

Citation from Tucker (1996)
Computer science remains a rapidly evolving discipline [...], which places considerable pressure on the [computer science and engineering] curriculum. The emergence of new tools, techniques and paradigms forces a continual re-evaluation of the topics covered and the pedagogical approaches used. Maintaining an up-to-date curriculum is particularly difficult for institutions because the pace of change in the discipline is so fast. As a result, the [computer science and engineering] curriculum quickly becomes outdated as the technology advances.

We take the same view, but nevertheless are convinced that there are basic topics of informatics which each engineering and science student should learn in an introductory course on informatics. These topics may be considered to be part of (general) literacy, but the fact that these act as fundament for further courses and continuing education is of larger importance.

Informatics as a new methodology should enter curricula in two different ways. First of all there should be some special 'inherently informatical' courses where basic insights, concepts, methods and skills are taught and practised. Secondly - and this is perhaps even of larger importance - informatics as a methodology should

enter practically all courses; with respect to both what is taught and how this is taught. It is of key importance that informatics as a new methodology gets 'under the nails' of all students. As for example reported by Kelemen (1996) the contents of informatics courses also very much defines the perception of informatics as a science and a new methodology. We are convinced that a distorted perception of the discipline may lead to the wrong understanding of the potentials and the applicability of informatics concepts, methods and tools.

The basic topics in informatics which should be mastered by all science and engineering students are:

- basic potentials and limits of the algorithmic approach;
- basics of efficient and feasible computing, including randomized and approximation methods;
- basic concepts and methods concerning security in information processing, including basic concepts and systems of cryptography;
- basic concepts, methods and pitfalls of systems (programs) validation and reliability;
- principles of data base (and expert) systems.

Students who have mastered these basic topics, should have general knowledge about the fundamentals of informatics and, expressed in a more goal-oriented way, should be able to make decisions on which of their problems to tackle by computer and how much effort this will cost. They should also be able to look for adequate software and hardware tools. As mentioned above we only refer to the minimum number of topics necessary to reach the overall educational goal. Much more advanced would be basic courses on simulation and visualization methods. These, however, should be integrated in the core study of the particular subject.

4.1 Specific approaches: fundamentals

Let us now discuss some specific ways to achieve the goals mentioned above. Of central importance is the notion of 'algorithm'. An analysis of the potentials and the limits of the algorithmic approach is needed. This may be done in various ways. We prefer an introduction by Random Access Machine (RAM). First, the principles of the von Neumann computer have to be explained, then a simple microcomputer can be presented. By the reduction of its set of operations and by neglecting the restrictions concerning the number and the size of registers, the desired simple form of a RAM can be obtained. And from this it is a natural step to a parallel RAM (PRAM) and to the introduction of some basic ideas of parallel computing and programming.

In order to grasp the right understanding of the algorithmic approach awareness of its limitations is necessary. What can and what can not be computed should be explained and various examples should be given. In this connection the notion of computational complexity, especially time and space complexity, should be introduced. (Here programs in a higher language might be more suitable than

Random Access Machines). The following hierarchy should be made clear: non-computable problems - unfeasible problems (not belonging to NP) - NP (-complete) problems - feasible problems (belonging to P). Especially NP-complete problems and ways how to cope with NP-completeness deserve attention. Students should also learn that one has to search for an efficient algorithm to solve a problem, and be aware of the difference between efficiency and effectiveness. The solution of difficult problems through heuristics, approximation and probabilistic algorithms should be explained.

Another important topic is problem specification. Examples may illustrate the advantages of formal descriptions for finding algorithms and developing of software. Students should learn that such a formalization will result in well-understood algorithms and associated data structures. Appropriate specification models are finite automata and regular languages, also allowing basic concepts of syntax and semantics of (programming) languages to be presented. Modelling of dynamic and concurrent systems can be done by Petri-nets. It should also be emphasized that there is no fully automated method for the design of formal specifications.

4.2 Specific approaches: software

The amount of software in technical products is rapidly increasing. All engineering and science students should have at least some basic understanding of problems and pitfalls in software development. They should have a basic notion of software specification, modular and hierarchical structures, correctness, validation, documentation and maintenance.

Because of the processing power which is available today, one might think that it is not longer meaningful to teach about algorithmic efficiency and software engineering because each correct program - also an unnecessarily complicated one - will terminate in a reasonable time. However, in all mature engineering areas efficiency and correctness are always of major concern. Students should at least be aware that only modular and hierarchically structured programs can be reasonably easily understood. Students should also learn two basic principles of good software design: a program has to be correct and should be easy to modify because frequent requirements for change are a fact of life. They should also know that software validation techniques should be used and that thorough documentation has to be delivered. A piece of software without detailed documentation should not be released. Also engineering students should be aware that nowadays more and more software is used in safety-critical applications.

Even well-validated and widely sold software packages can often only be used by graduates who understand their technical merits and aims. The corresponding knowledge will be part of informatics methodology taught by the departments of engineering or science. Graduates should be able to make plausibility estimations about the correctness of the results obtained using algorithmic methods and

software packages, and they should be acquainted with the problems of numerical stability.

As mentioned before students should have some experience with the programming of numerical problems. But they should be aware that nonnumerical data processing has at least the same importance and that processing of big amounts of data brings additional problems. The principles of data bases should be presented and also how expert systems work and what their limitations are. Security of information exchange and storage is another key issue in this era of modern communication and students should learn some basic concepts of cryptography and its protocols and of information security. Ethical questions should be posed and discussed, not as a separate topic, but integrated in the whole course. There are no definite answers to these problems , rather it has to be pointed out that ethical problems exist.

5 COMPARISON WITH OTHER APPROACHES

It may be useful and interesting to make a detailed comparison of our proposals with the topics which have been suggested in various other undergraduate curricula in informatics, but this would go beyond the scope of this paper. Let us just make a few remarks.

From the nine subject areas in Denning *et al.* (1989) only a minor part is covered by these recommendations. We think that some of the other ones should be incorporated in the curricula of the various disciplines. For example, electrical engineering students who have to design or maintain computers, should have much deeper knowledge in informatics than those students who only use computers as a tool. Architecture and programming languages are indispensable topics for mechanical engineers who deal with the development of embedded systems. For some of them a course on artificial intelligence and robotics could play a crucial role. A lot of work of civil engineers is done with graphics tools; for them more knowledge about computer graphics and human-computer communication would be helpful. And for physicists numerical and symbolic computation will be important.

All these topics should be taught in special courses, not being part of the proposed 'minimum curriculum' which should only contain the most important subjects common for all engineering and science students. This also holds for other topics not included in the list above, e.g. for such an important subject as parallelism. Here we have to restrict ourselves - according to Tucker *et al.* (1991) - to the goal to 'prepare students to apply their knowledge to specific, constrained, problems and to produce solutions. This includes the ability to define a problem clearly; to determine its tractability; to study, specify, design, implement, test, modify, and document that problem's solution; ...'.

6 CONCLUSION

The suggestions in the preceding sections should be considered as a collection of topics which can be viewed along three stages:

- problem analysis;
- algorithm design;
- program realization.

It seems appropriate to start with easily described every-day problems, to demonstrate the usefulness of formalization and then to develop algorithms. In our experience, graph theory problems and algorithms are very instructive for this. Many of such problems and their solutions are easy to grasp and, moreover, very useful for applications. Our suggestion is to connect theory, software and applications in the educational process (see for example Gruska and Vollmar (1997)).

To present the core material in a not too compressed form about 60-80 hours are necessary. In addition about 25-30 hours for laboratory work should be planned. Not included is the time needed to teach the material mentioned in the section on the propaedeutic. More important than the overall number of teaching hours is the atmosphere in which the topics are taught and the approach to informatics which professors of engineering and science show. If they are convinced that methods and tools of informatics are valuable for their disciplines and if they are passing this attitude on to their students, good results may be obtained even in a shorter time.

7 REFERENCES

Denning, P. *et al.* (1989) Computing as a discipline. *Communications of the ACM*, **32**, 9-23.

Gruska, J. and Vollmar, R. (1997) Towards adjusting informatics education to the information era, to be published in a book from Springer-Verlag.

Kelemen, C. (1996) *Public understanding of Computer Science and first courses for non-majors.* http://www.bowdoin.edu/allen/sdcr/kelemen.html, World Wide Web.

Krüger, G. (1987) Informatikgrundlagen in der Lehre an Hochschulen, in *Informatik-grundlagen in der Lehre* (ed. Bundesminister für Bildung und Wissenschaft), 105-113.

Tucker, A. B. (1996) Strategic directions in computer science education. *ACM Computing Surveys*, **28**, 836-845.

Tucker, A. B. *et al.* (1991) A summary of the ACM/IEEE-CS joint curriculum task force report: Computing curricula 1991. *Communications of the ACM*, **34**, 68-84.

8 BIOGRAPHY

Roland Vollmar was born in 1939. He received his diploma in mathematics from the University of Saarbrücken (Germany) in 1964 and his Dr.-Ing. degree from the University Erlangen-Nürnberg in 1968. Since 1965 he was affiliated with several institutes for computer science and with an industrial company. From 1974 to 1989 he was full professor in theoretical computer science at the Technical University of Braunschweig and since 1989 he has a same position at the University Karlsruhe. His main research interests include parallelism, especially cellular automata. He is author and. co-author of three books on these topics and of numerous articles.

Jozef Gruska graduated in mathematics in 1958 at the Comenius University in Bratislava and received his Ph.D. in computer science from the Slovak Academy of Sciences in Bratislava. He is professor in computer science at Masaryk University in Brno, Czech Republic, and has had a variety of long-term visiting research and teaching positions at universities in Europe, the USA and Africa. His research interests are: scientific computing, theory of automata and descriptional complexity, parallel automata and systems, and foundations of informatics. For a long time he has been heading the research seminar in Bratislava. He is the author of various books and curricula. His main international activities are: council member of EATCS (1985-91), country representative in IFIP-TC2, member of the advisory board of IFIP-TC12, chairman of the IFIP Specialist Group on Foundations of Computer Science(1989-96).