

Computer Science as a Core Subject in Engineering Education

Helmut Ketz, Karlheinz Hug
Fachhochschule Reutlingen - University of Applied Sciences

Published in: Peschges, Klaus-Jürgen (Ed): „Sharing Experience to Increase Internationalization and Globalization in Engineering Education“, International Conference, Fachhochschule Mannheim - University of Applied Sciences, 17.-19.9.1998, Conference Proceedings, S. 429-433

Abstract

Nowadays many classic domains of engineering disciplines are "undermined" by programmable electronic circuits and software components forming so-called "embedded systems". The challenging question for engineers is: Is it sufficient to teach a little bit of data processing skills in using a computer as a tool, or is it necessary to continue in teaching computer science as an engineering discipline and a mean for mastering complexity? A look at typical engineering curricula shows - at least at German universities of applied sciences - that some courses on some well established programming languages is almost all engineers get to know about computer science. Still there is a knowledge transfer gap between computer science and engineering disciplines. This article describes the current situation and approaches in integrating computer science as a foundation in engineering disciplines - here electronics and automation. We focus on experience at a German university of applied sciences.

1 Introduction

Both authors teach computer science in engineering departments at a university of applied sciences (Fachhochschule) in Germany. Consequently the background of the article draws from the typical situation at this kind of university. Normally, our students finish their studies after about eight to ten semesters with a diploma degree. In this paper, we are concerned with engineering studies completing with the diploma. Nevertheless, we believe that our observations, challenges and approaches to didactically master the technological changes are not unique in today's technology community.

In Germany computer science is called informatics (Informatik). This notion is not restricted to the computer as a machine, but emphasises the notion of information and its role in social-technical relations such as the design of production processes or co-operative work. In this article, we use the terms informatics and informatician (computer scientist) where appropriate.

It is widely accepted to distinguish four branches of informatics: theoretical, technical, practical and applied informatics. When discussing informatics education for engineering students, the technical and practical parts are most relevant. Software engineering, considered as part of practical informatics, has the strongest relations to other engineering disciplines.

2 Change in Technology

Developing automated or embedded systems requires increasingly profound knowledge of information technologies (IT) in system requirements analysis, specification, architectures, design and implementation. Typical applications are highbay warehouses, automated assembly lines, medical surveillance equipment. Generally they integrate programmable controllers, robots and numerically controlled equipment. They may also be based on sensors and actuators, which formerly were typical domains of "pure hardware".

The same holds for classical engineering tasks such as management and maintenance of application systems. They deal with IT logistics based on hundreds or even thousands of computers connected by networks. A good understanding not only of physical processes is needed, but also the capability to handle information-based processes.

New offerings in engineering faculties like automation, mechatronics or cognitive science reflect this rapid development. Among them are lectures in microprocessor technologies, assembler programming, operating systems, networking or artificial intelligence.

On the other hand, informatics, once a child of electronics engineering and mathematics (at least), has grown into a science with its own subjects and methodologies. In particular, programming evolved from an art - as it was seen in the sixties - into an engineering discipline called software engineering. Formerly, software concepts and methodologies were often based on heuristics or centred around some idea as to how to solve a certain problem class. Recently developed software concepts and methodologies are much more theoretically founded and intended to solve a entire range of practical engineering problems.

For example, the concept of data abstraction is based on a mathematical theory using algebraic terms. Coincidentally, it provides a key to many practically usable technologies such as object and component software systems or distributed and client-server architectures [5].

3 Informatics as a Minor Topic in Engineering Curricula

Unfortunately, computer science education in engineering faculties has not yet reached the level of maturity of informatics. Often it remains restricted to teaching programming-in-the-small with a specific and trendy programming language. However, this approach does not address the challenges of complex integrated systems involving programming-in-the-large. To this end, students need a solid foundation of software concepts and methodologies [1].

Some teachers are not well educated informaticians; some have learned autodidactically how to program and use programming tools and libraries. Sometimes such a lack of education and professional background is honoured as "bound to practice". From such a viewpoint, computer scientists appear to be "too much theoretical" because they prefer modelling techniques and conceptual approaches to problem solving and software construction instead of immediate trial coding and error monitoring with debugging tools. This approach is reflected in lectures. The level of quality with the "self-taught trial-and-error" kind of education is hardly suitable for programmers, let alone engineers [7].

The weakness of the situation is visible. Software construction is a serious business, it should be taught and performed on a professional basis. Looking back in the history of data processing, we see a similar development in the late sixties and early seventies. This definitively means: The problem is recognised among IT professionals and in the literature. Why is this knowledge ignored instead of being used in today's discussion about engineering curricula? Is it because the speed of proliferation of computing equipment is much faster than the learning curve of human beings?

4 Industry's Demands

We are in close contact with the local industry as well as global players. When graduated, our students will get their jobs there. We want to prepare our students so that they can find their own way and take over tasks that must be performed responsibly. We try hard to offer a good mix of fundamental capabilities and industry's demands in our curricula. However, which skills is industry looking for? This is not a trivial question!

Whenever we examine job offerings for engineers in the weekend editions of our newspapers or on the WWW, we notice that the printed job requirements differ tremendously from the qualifications asked by the hiring managers in the engineering and IT departments.

Typically, job offerings ask for experience in specific programming languages, release-dependent know-how in IT products (e.g. databases, communication networks), tool-handling skills and so on. Obviously, the personal department is looking for someone who will be immediately productive for the company starting at the first day on the basis of today's products (fit for the job).

Hiring managers generally search for long-term employees and they are also willing to invest in their engineers. They look for technical competencies as well as for social and knowledge transfer competencies. Ranking at the top are interpersonal communication skills, abstraction and analytical skills, technical skills for the job and systems engineering skills. Managers tend to rely on the capabilities of academically educated engineers to acquire the skills of the offerings in a short time on their own (competent for the job).

The approaches for hiring sketched above do not fit together well. This conflict is reflected in discussions on campus about forming curricula. Our viewpoint is clear: Engineering education should aim at competence, not merely at fitness for the job.

Latest numbers of the "Bundesanstalt für Arbeit" say: There is a demand in Germany for about 22.000 informaticians per year. German universities (including those of applied sciences) only output about 11.000 informaticians per year. Actually, there is need not only for system developers but for programmers in particular. On the other hand, up until now there is no proper professional education for scientific/technical programmers in Germany. Only recently have some IT apprenticeships below the bachelor level been introduced.

For historical reasons, many engineers work as programmers - without a fundamental education in informatics. They may meet several preconditions for their job: a strong mathematical background, a good understanding of engineering techniques, a profound experience with technical application systems. Maybe that is why industry pays engineers' salaries for people dealing with programming language problems most of their time. However, engineers should work on the qualification level of engineers, not on the level of programmers. So, what can we do about it?

5 Common Interests of Engineering and Computer Science

We can identify common roots of engineering disciplines and computer science, but differences as well.

- Engineering has a long history, numbering in hundreds of years. Computer science is still very young, but it is developing into an engineering discipline.
- Engineering has a strong physical and mathematical background and deals mainly with quantities. Computer science has a mathematical background too, but is also strongly bound to semantics and therefore deals with qualities rather than quantities.
- Engineering is mostly based upon formal mathematical calculus. Computer science often uses semiformal models and notations. Moreover, its models are rarely based on numerical, but on discrete mathematics.
- There is a broad agreement in the engineering community about how to apply engineering methods. In the IT-world, no such an agreement exists. In particular, there is no commonly accepted answer to the question which qualifications one needs for being a good programmer.
- In both disciplines, notions like system, architecture, component, interface, process are of importance.

For a discussion of such issues see e.g. [8].

Nowadays, formalised numerical processes in engineering are automatically calculated by computers. The slide rule is exhibited in museums. The process of modelling systems has become more and more important in the daily life of engineers. This change is reflected in the curricula of engineering faculties [4].

In integrated systems software is always a subsystem - though a very important one. So there is a need to teach computer science as an engineering discipline. This is a challenge far beyond that of programming-in-the-small, using tools and libraries on a given architecture (e.g. programming graphical user interfaces). Designing architectures is one generic task of an engineer.

Why should computer science become a core subject in engineering disciplines?

- First of all, an engineer must be able to understand an informatician. Engineers and informaticians have to co-operate in project teams. To this end, they need a common expert language. Modern engineers should bridge the gap that has opened between classical engineers and informaticians.
- An engineer should have enough educational background to specialise in the software engineering area. To this end, he must at least be able to read the relevant computer science literature.
- There is a shortage of system developers in the market. As long as computer science departments at universities cannot satisfy industry's needs for software engineers, engineering departments should help.
- Application know-how is becoming more and more important for systems developers.
- Engineers have a strong mathematical background and so they are excellently prepared for developing scientific and technical systems, e.g. embedded systems.
- Classical mechanic- and electronic-bound systems are becoming more intelligent today by integration of software components. There is a need for common architectures of hard- and software.
- Configuration, management and maintenance of such systems require a good understanding of software

systems and architectures.

- System modelling approaches in engineering converge with system modelling in computer science.
- We need comparable quality measures for analysis, specification, modelling, design and implementation in hard- and software in order to build complex systems with overall, consistent quality levels.

So the question arises: What are teaching topics that will be valid for quite a while after graduation?

6 What are Fundamentals in Informatics?

Education aiming at competencies for the job has to train abstraction as well as modelling competence, including results of cognitive science referring to problem solving or creativity. The engineer, dealing with integrated systems, is a bridge constructor (pontifex) for the semantic gap between the application and the technical system. There he needs interpersonal communication competence, a high level of abstraction capabilities and skills for transforming different notations - describing systems on different levels of abstraction - from one to another. Those notations may be strictly formal, semiformal or even informal.

Within this context, a programming language is but one of many means for coding designs and cannot be the core business of teaching. Informatics deals foremost with structures. Thus, if we teach fundamentals we are concerned with questions like these:

- What are algorithms and their properties? Given an algorithmically solvable problem, how can we construct an algorithm?
- Recursion and self-references - why and where do we need them, how do we use them?
- How do we cope with complexity? Modularization - what are appropriate units for decomposing a complex system?
- What makes data abstraction a key concept of software technology?
- Static and dynamic structures - what are the pros and cons? When and how do we use them?
- Sequential and concurrent structures - what challenges does concurrency offer?
- Object-orientation - what are its key concepts, which problems does it tackle, which does it leave open?
- Component technology - how does it cope with some limitations of object technology?
- Software quality - how do we achieve reliable, reusable and extendible software products? How can we construct software that meets functional as well as quality requirements?
- How do we bridge the gap between a problem domain and a set of problem solving mechanisms?
- How can we manage software development as an intellectual, creative and co-operative process involving people with different views and interests?

7 How Do We Face the Challenges?

Generally, our departments consider informatics as a core discipline in their curricula and try to integrate engineering and computer science by lectures like system engineering, project management, creative problem solving, by practical projects or by diploma thesis. How can we lay the fundamentals of computer science during the first semesters? Here we would like to point out some of the aspects.

We start holistically with object-oriented approaches of modelling small systems, such as dealing with algorithms and data structures. We train "thinking in structures" by changing levels of abstraction continuously in order to achieve flexibility. To this end, we use different notations. One of them is a programming language of adequate level. We not only stress functional views but also relations between modules and components. Constructive aspects are trained in practices on the computer. Here we encourage teamwork. We deal with software quality, methods of reducing complexity and basics of programming languages (grammar, compilation, run-time properties). Since we try to integrate computer science with engineering from the beginning, many of our examples are technically bound.

In order to reach the above mentioned goals we need a simple, easy-to-learn programming language, rather

than one that is binding all the time of our lectures for dealing with cryptic code constructs or tips and tricks or even a huge pack of manuals with several hundred pages. We have chosen Oberon, a lean object-oriented programming language designed by Niklaus Wirth and Hanspeter Mössenböck [6, 9]. Our experience is satisfying, including our students' affirmation, proved by the evaluation of our lectures [2, 3].

8 Conclusion

There is a need for integrating computer science in engineering education, especially in electronics, mechatronics and automation. The integration should be achieved by rigorously keeping a consistent and adequate level in teaching. This requires co-operation and mutual understanding of all participants. In particular, computer science has to be accepted as an engineering discipline with common as well as specific methods of abstraction. Integrating informatics in engineering disciplines definitively leaves the narrow approach of "informatics is programming" behind. With a wider approach, the offerings of informatics to engineering can be exploited in favour of future applications with higher quality.

References

1. Freytag, J.: *Empfehlungen der Gesellschaft für Informatik für das Informatikstudium an Fachhochschulen*. Informatik-Spektrum 19(1), 20-32 (1996)
2. Hug, K.; Ketz, H.: *Objektorientierung mit Oberon-2 in der Ingenieur-Grundausbildung*. Informatik-Spektrum 20(6), 350-356 (1997)
3. Ketz, H.; Hug, K.: *Informatik-Grundausbildung für Ingenieure - Hochschuldidaktische Betrachtung und Erfahrungsbericht*. In: Claus, V. (Hg): „Informatik und Ausbildung“, GI-Fachtagung 98, Stuttgart, 30.3.-1.4.1998, Berlin: Springer 1998, S. 52-62
4. Lee, E.A.; Messerschmitt, D.G.: *Engineering an Education for the Future*. IEEE Computer 31(1), 77-85 (1998)
5. Meyer, B.: *Object-oriented Software Construction*. Englewood Cliffs, New Jersey: Prentice Hall 1997, 2. ed.
6. Mössenböck, H.: *Objektorientierte Programmierung in Oberon-2*. Berlin, Heidelberg: Springer 1994, 2. Aufl.
7. Nievergelt, J.: *Was ist Informatik-Didaktik? Gedanken über die Fachkenntnisse des Informatiklehrers*. Informatik-Spektrum 16(1), 3-10 (1993)
8. Ott, H.-J.: *Das "Ingenieurgemäße" am Software Engineering*. Softwaretechnik-Trends 14(1), 31-37 (1994)
9. Reiser, M.; Wirth, N.: *Programmieren in Oberon. Das neue Pascal*. Bonn: Addison-Wesley 1994

Authors:

Prof. Dipl.-Inform. Helmut Ketz, Fachhochschule Reutlingen - University of Applied Sciences
Fachbereich Automatisierungstechnik - Faculty of Automation Engineering
Alteburgstr. 150, D-72762 Reutlingen, Germany
Phone: (+49) 7121 271-349, Fax: (+49) 7121 271-605, E-Mail: Ketz@at.fh-reutlingen.de

Prof. Dr. Karlheinz Hug, Fachhochschule Reutlingen - University of Applied Sciences
Fachbereich Elektronik - Faculty of Electronics
Federnseestr. 4, D-72764 Reutlingen, Germany
Phone: (+49) 7121 341-109, Fax: (+49) 7121 341-100, E-Mail: Karlheinz.Hug@fh-reutlingen.de