Small Is Beautiful, Isn’t It?
Contradictions in Software Engineering Education

Péter Hanák, Zoltán László
Hanak@inf.bme.hu, h4245las@ella.hu
Faculty of Electrical Engineering and Informatics,
Technical University Budapest
H-1521 Budapest, Hungary

Abstract
Staff and students have recently faced the latest reforms in informatics education at the Section of Technical Informatics, TU Budapest. In this paper, partly post factum, we pose ourselves such questions as

• which topics of informatics, and in particular of software engineering, ought to be taught,
• how these topics should and could be taught,
• who and how will present these topics,
• how to convince students of their importance and usefulness (i.e. how to motivate students),
• what jobs will be open for software engineers in Hungary or in other countries,
• what are the skills a university student should acquire,
• what is the optimal ratio of theory and practice, etc.

We do not promise the answers to these and many similar questions. However, we do try to reveal contradictions in software engineering education in a small country, partly by comparing our problems to those discussed in the literature, and partly by presenting our experiences and approaches at TU Budapest — in the hope that it will trigger vivid discussions at the conference on Shifting Paradigms in Software Engineering in Klagenfurt.

1 Introduction
Informatics (as it is called in Continental Europe) or computer/computing science (as the American/British traditionally call it) is related to mathematics, engineering and management. Depending on the School where it is taught one of its aspects is emphasized. Nonetheless, in the curriculum a proper balance is desirable. In Section 6 the newest curriculum of technical informatics at TU Budapest, centred around software engineering (SE), is presented, and the sequence of courses that determine its SE content is discussed, with an eye kept on this balance.
Engineering is defined in [10] as ‘creating cost-effective solutions to practical problems by applying scientific knowledge to building things in the service of mankind’. SE is claimed in the same paper to be more ‘a statement of aspiration’ than a ‘description of accomplishment’ because of the ‘lack of widespread routine application of scientific knowledge to a wide range of practical design tasks’.

It’s not at all easy to determine the necessary content of a degree programme in informatics and in particular in SE. Beside personal ambitions, local expertise and available infrastructure the immaturity of the subject causes most difficulties: the 40-year history of computing education is the history of permanent shifting, triggered by technological changes, from technical peculiarities to higher-level concepts and solutions.

By now, programming-in-the-small is more or less well understood, and the educational community has the necessary skill in teaching the widely accepted, sound principles of algorithmic and data abstraction. On the other hand, programming-in-the-large is far from being established: it is a field of discussions and beliefs. After all, small is beautiful, isn’t it?

While we still strive for theories, methodologies and tools necessary to create huge but correct and secure software systems one wonders if the well-known educational difficulties are only due to this lack of knowledge, or if they are of intrinsic character. That is, we have to face the problem of teaching complex things while being restricted in resources, time and prerequisite knowledge. Although this situation is not unusual in the engineering education we again have to ‘reinvent the wheel’ in the SE field — as it occurred e.g. with structured programming.

Then, we should ask ourselves whether the idealized practice of system design can be abstracted and taught at all, or only some well-sounding principles and slogans can be collected and presented in the classroom. The authors, graduated in electrical engineering and gained practice in the design of medium-size hardware/software systems, have the impression that systematically only small-scale design of digital systems is and can be taught. It would be interesting to study other, more mature and less dramatically changing fields of engineering (e.g. civil or mechanical engineering) in order to reveal the similarities and the differences, and to see whether there are any general methods applicable also to SE.

Education is usually told to be the art of concealment, but it could well be called the art of selection, i.e. of choosing things worth to know. Very frequently, we teach nonessential and unimportant topics just because they are well known and easy to teach.

The dilemma was also admitted during a workshop at Brown University in 1990 [3]: many invited speakers questioned even the widely accepted contents of introductory programming courses describing them as nonrelevant. Further, ‘it was a humbling experience to see that after twenty-five years of teaching computing in major universities, we still don’t know how to do it’ [4].

In the first paragraph we used the term ‘technical informatics’. In Hungary, it has been used since 1991 meaning informatics education at universities and colleges of technology, in contrast to ‘theoretical informatics’ being or to be taught at universities of natural sciences, ‘econometrics’ at universities of economy, and ‘library informatics’ at universities of liberal arts, etc. In the courses of technical informatics ‘much more attention is given to the hardware aspects of information systems than anywhere else’ [8], and to its engineering character.

2 Contradictions and other problems

Here is a list of the most striking contradictions that deeply affect engineering education in general, and SE education in particular. We suppose they are understandable without