

The Software Engineering Curriculum at the University of Stuttgart*

Jochen Ludewig Ralf Reißing

University of Stuttgart
Institute of Computer Science
Breitwiesenstr. 20-22
D-70565 Stuttgart, Germany

{ludewig,reissing}@informatik.uni-stuttgart.de

Abstract. *At the University of Stuttgart a new curriculum called Software Engineering was launched in October 1996. It is offered by the Computer Science Department and supplements the standard curriculum in Computer Science started in 1970.*

This paper first gives an introduction to German universities in general. After this the reasons for starting a new curriculum, its goals and its features are discussed. Finally, the current state and the next steps are outlined.

1 Universities in Germany

The education system in Germany differs significantly from the systems in most English speaking countries. Therefore, some basic facts about German universities may be useful for most readers. The following statements hold for computer science (and for most other majors):

- Those who have finished school at the highest level (about 22%) are accepted at any university; there are no entrance exams.
- There is no distinction between undergraduate or postgraduate students, because there is no degree below the diploma-level. Note that a diploma is in

Germany a generally recognised, legally protected degree that all engineers and most scientists obtain at university. It is equivalent to a master's degree.

- The examinations are not part of the courses; students may register for exams anytime they want to. Hence, many students tend to postpone examinations.
- Students get comparatively little guidance and no continuous attention from the faculty. When they get into trouble, but do not ask for help actively, they will not get any support. The dropout quota is high (about 50%), partially because many students have jobs which keep them busy outside university.
- Studying at a German university is (almost) free.

Compared to the conditions in most other countries, the German system is a challenge for the students. They have to organize their studies and they have to decide what is important. They must be independent and responsible. Those who manage to complete their studies have learned a lot. For many others, a rigid curriculum with less freedom would be better.

2 Computer Science in German Universities

All German curricula in computer science are based on the same structure (Fig. 1). After

*An earlier version of this article has been submitted to *IEEE Software*.

Proceedings of the Fourth International Workshop on Software Engineering Education, May 23, 1997, Boston, MA, USA (in conjunction with the 1997 International Conference on Software Engineering).

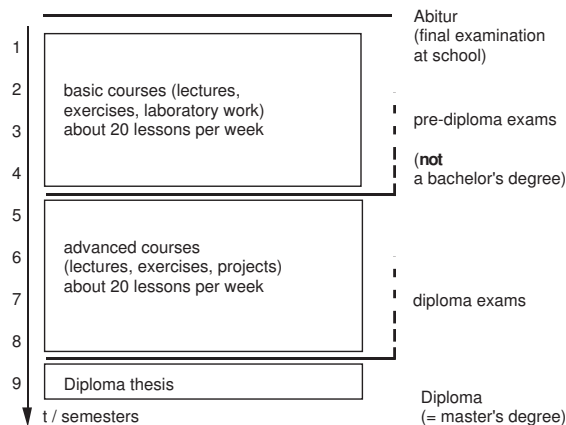


Figure 1: General schema for an engineering or science curriculum in Germany

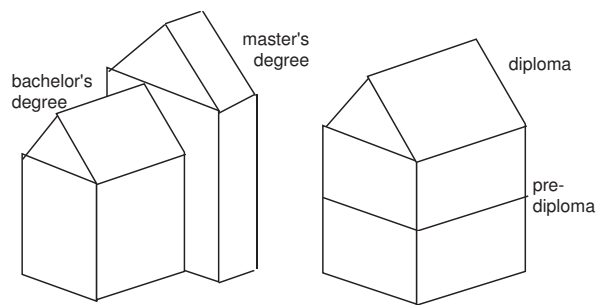


Figure 2: The difference between a bachelor's degree and a pre-diploma

two years of studying there is a set of examinations called “Vordiplom” (pre-diploma). After another two years, students should have finished all their exams. The final semester is to be spent on the diploma thesis. Then, after 9 semesters, the student will receive a diploma, which is equivalent to a master's degree. Note that the average career of our students is far more chaotic than Fig. 1 suggests. Deviations occur due to individual reasons, sometimes also to organizational deficiencies in the university.

The pre-diploma examination is only an intermediate test, which permits those who passed to the second half of the curriculum (see Fig. 2).

It is not a bachelor's degree, in fact, it is no degree at all, and it has no meaning on the job market. Nobody enters a university with the intent to leave after the pre-diploma exams.

Therefore, the first and second year don't have to constitute a complete education. They contain all the basic and theoretical courses which provide the foundations for more specialized courses taken in the third and fourth year.

3 From Computer Science to Software Engineering

In many universities, computer science is taught like a mathematical science, i.e. as a collection of theories, analytical methods and formal notations. These topics are obviously important for our profession. But most of our graduates find jobs where they build new software systems or modify existing ones. They are doing the same kind of work engineers do, only with different goals and materials.

Therefore, for them the constructive aspects of computer science are more important than the analytical aspects. In other fields this distinction has eventually ended in a happy divorce: Maxwell's equations are fundamental both for physicists and for electrical engineers. But the physicists use them in order to understand, while the electrical engineer applies them (usually in some simplified shape) in order to build something (see Fig. 3a)

In computer science the situation is not yet mature for such a divorce. But our new software engineering curriculum (SEC) is an attempt to give an example of what such a “constructive computer science” might look like (Fig. 3b). The SEC is still full of compromises, mainly because there are simply not enough faculty members to start from scratch. But the general structure seems to be sound.

4 Outline of the Software Engineering Curriculum

Fig. 4 shows a (radically simplified) overview of the SEC. Provided there are no special obstacles, students will usually try to follow this schedule for the first four semesters, i.e. until the pre-diploma. After this they choose their

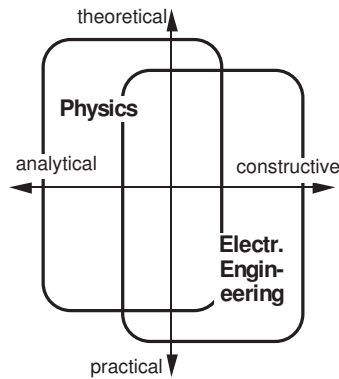


Figure 3a: Electrical engineering versus physics

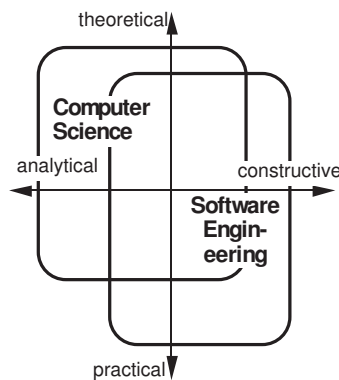


Figure 3b: Computer Science versus software engineering

individual paths. The numbers in the boxes are quantities; “1” stands for “one lesson of 45 min per week in 15 weeks of the semester”. “L” stands for “lecture”, “E” for “exercise”, “P” for “practical work”. The figures on the right hand side indicate the workload in each semester (always around 20). Shaded areas denote project work, including the diploma-thesis that should be considered as a one-person-project.

5 A new Approach to Theoretical Computer Science

New courses in theoretical computer science are essential for the SEC.

The education in mathematics, theoretical computer science (TCS) and other theoretical areas, as offered by our universities, is credited highly. While no one in industry wants us to

sem.	mathem. and econom.	theor. CS	introduction to informatics	technical CS and exercises	
1	5L 2E Adv. Math. I	3L 2E Logic	4L 2E introd. I	4 P Programing	2E English
2	5L 2E Adv. Math. II	3L 1E Theor. CS I	4L 2E introd. II	3L 1E Techn. Inform. I	6L Fundamentals of technolog.
3	4L 2E Economics	2L 1E Theor. CS I	4L 3P introd. III	3L 1E Techn. Inform. II	
4	2L 1E statistics	3L 1E Theor. CS I	4L 2E introd. IV	4 P Basic SW project	
5	16 Main courses (4 out of 5, always including SE)	14 Project 1 in the informatics dept.	12 Application area		
6					
7		14 Project 2 with legacy software	14 Project 3 in the application area		
8	10 Arbitrary courses in CS				
9	20 diploma thesis				

examination or proof: certificate written or oral examination mark on performance

Figure 4: General outline of the SEC (software engineering curriculum)

extend these courses, very few want them to be reduced. Sound theoretical foundations are approved.

Most students do not like formal theories and only learn them to pass their exams. Afterwards, they soon forget most of it. Therefore, most of them will never learn how practical and useful theory is. When questions are not asked in a formal context, but are taken from an application area, very few students can identify the underlying structures they learned in TCS. On the other hand, they are hardly able to name a single practical usage for any of the theories. Theory and practical work seem to be different areas having practically nothing in common.

We are going to change that. We want neither more nor less theory than we have in the computer science curriculum, but a different TCS, a “practical TCS”. What could such a TCS look like?

Traditional lectures and books contain many theorems, each one followed by a proof, not by its typical or most important applications. Many proofs demonstrate that something is not possible. This is in sharp contrast to the actual needs: except for some professors and researchers, the only purpose of a theorem is its application; it is enough when a student has seen a few proofs and understands the principle of proving. There is no need to prove dozens of theorems. But there is an urgent need to show for every theorem its particular power in practical applications. Theorems whose power cannot be shown should not be treated.

In software engineering TCS is not an end in itself, but a useful aid in solving real problems. Therefore, students must attend all necessary courses in TCS before finishing the pre-diploma.

6 Projects and the Application Area

There is no doubt about the importance of practical experiences. This holds, of course, for all engineers. But in software engineering, practical experience is not always as beneficial as it should be, because many companies demonstrate poor software engineering. Under such circumstances all a student can learn is that software engineering is hard, often too hard for those who never learned it properly. When graduates are hired for their first job, they are often not expected to continue learning, but to teach their colleagues.

If we want to make sure that students get a chance to apply what we are teaching, thus strengthening their new knowledge, we must offer a lot of practical work within our curriculum. The most important type of such work are student projects.

Since our students have – at least – two years between pre-diploma and diploma thesis, we require them to participate in three different projects. These overlap, because they take twelve to eighteen months each. Projects are performed in groups of six to ten students.

What is the most important aspect of such a project? Is it the goals? The organization? The people? The time schedule? Well, all these components are important. But, at least for a project in software engineering, most important is the existence of a customer, who defines success and failure. An average customer does not understand the software people, and vice versa. Customers often change their minds, are not willing to read formal descriptions and argue about whether or not the user interface meets the requirements. Any project without such a customer is far too unrealistic to be useful. The worst – but fairly common – situation occurs when students are their own customers. How can they learn to communicate about requirements with a customer who does not speak computer science? Who teaches them to deal with unexpected changes? Where do they experience that the primary purpose of a piece of software is to please the customer, not the developer?

Project 1 is an academic project under strict project management organized within the Computer Science Department. The customer will usually be a professor. Tutors make sure that students follow the process and reach their goals. After project 1, the participants have learned that there is a way – not always a smooth one – that leads from the problem to its solution. They have experienced the effects of poor documentation and insufficient communication and they have seen software quality assurance from both sides. They have made presentations and have written reports. Additionally, in contrast to projects in industry, they have had the opportunity to reflect and discuss all difficulties.

In the second project existing software is the additional hurdle. We have no knowledge of any computer science curriculum where existing software plays a significant role. But most tasks in the world of software are concerned with existing systems. Therefore, project 2 includes existing software, which may be re-engineered, replaced, modified or supplemented. In addition to the duties mentioned for project 1, students are involved in the orga-

nizational tasks like planning and project management. Since project 2 necessarily deals with a real problem, there is also a real customer, who should pay for the project in order to stimulate his or her attention.

After the pre-diploma, every student has to choose an application area. As the name indicates, an application area is a field where software is applied as, for example, robotics, traffic planning or economics.

When a student chooses, say, robotics, we do not want him or her to become a second class engineer. We want the students to leave the comfortable playgrounds they have enjoyed for years. We send them to a different world, where they have to understand the problems of those who do not understand the software solutions.

In project 3, the application project, students must combine their abilities in software engineering with their knowledge about the application area. This project will usually be the one which is closest to industrial reality. In these projects the computer science department is involved only when the project is planned and outlined, because all projects must undergo some inspection before they are officially announced.

7 Other Features of the Software Engineering Curriculum

We believe in programming as an important craft that is, and remains, very important in all areas of software engineering. Therefore, we have added two new programming courses (in the first and third semester). These courses prepare the students for the bigger projects.

Business economics is not only an important application area of computer science, but also vital for producing, buying and comparing software. We have integrated a course in business economics in the third semester.

The English language is extremely important for those working on software: Many manuals, almost all international conferences, all

programming languages are in English. We require our students to participate successfully in a TOEFL (Test of English as a Foreign Language; they must achieve a minimum of 520 points).

In order to collect some basic insight into the terminology and methods of engineering, students should attend some introductory courses which are given in the engineering departments.

Many problems in software engineering are hard to understand for those who have never experienced life outside university. Therefore, we require our students to spend at least four months on jobs in industry. We do not have detailed requirements about the type of work our students do; it is the social experience that counts most.

In a recent investigation [1], we found that disorientation, lack of information, long delays and dropouts are strongly connected. Tutoring and working groups from the very start, early examinations with a couple of tough ones after the second semester and projects that keep the students close together are the measures to make more students finish their studies successfully and in less time.

8 Where are we today?

The concepts of the SEC were completed in March 1996. They passed through all levels of our university within three months, which was incredibly fast. The Secretary of Education agreed in July 1996 and allowed us to enroll up to 60 students. Such a limitation is unusual in computer science, but the SEC is what is called a “Modellstudiengang”, an experimental curriculum. When the first students finish their studies, i.e. after five to six years, the achievements will be examined, and the SEC is either terminated or changed into an ordinary curriculum (possibly with some modifications). In that case, we will probably shift the traditional curriculum towards a more analytical profile, thus offering two truly complementary curricula.

Since we could not publicly announce the SEC before July 1996, we did not expect to find many students, so we didn't prevent over-subscription. But when the SEC started in October 1996, 76 students had enrolled, so we had exceeded our limit.

The SEC emerges like a road being built for a vehicle that is steadily moving towards the construction site. A small committee makes sure nothing is missed, and the necessary courses are planned in detail. Students from the SEC and senior students from the computer science curriculum participate in this.

Response from industry is enthusiastic. Projects, team work, better training in presentation and communication, courses in business economics and English, experience with legacy software: this is music to the ears of those recruiting employees for jobs in software engineering. The "Foundation for Science in Germany", the national sponsoring organisation of the German industry, has awarded a prize of 75,000 DM (about \$45,000) to the SEC, a most welcome support.

Many universities in Germany have asked for information about the SEC; some have started preparations for launching a similar curriculum. One of the most convincing arguments is that 209 students enrolled at our department in 1996, compared to 129 in 1995. No other computer science department in Germany has had a similar boom.

9 Acknowledgements

The software engineering curriculum was only possible with the support of Volker Claus, Andreas Reuter, Rul Gunzenhäuser and other colleagues in the Computer Science Department in the University of Stuttgart.

References

1. Claus, V. (ed.) (1996): Evaluation von Studium und Lehre 1995/96. Bericht der Fakultät Informatik, Universität Stuttgart.