We Do It Ourselves

J.J. van Amstel
Philips Research, Information and Software Technology
Education and Training
Prof. Holstlaan 4
5656 AA Eindhoven
The Netherlands

Abstract

In Philips there is a need for many competent software developers. Not for everyone in the company software development is a real engineering activity. They claim that there are no fundamental theories underlying software and software construction. On the other hand, only a few universities offer a curriculum in software engineering, most curricula deal with computer science.

Since ten years we have had in Philips our own education and training programme for software engineers. We are now in the stage of formulating new specifications for this programme. We hope that a (large) part of the new programme will be filled in by universities.

1. What is it?

The importance of embedded software is still increasing. The functionality of all kinds of products is more and more dependent of the software contained in these products and this software content is growing fast. The software content of Philips consumer products doubles approximately every two years.

In Philips a considerable number of people, formerly working in a non-software environment, are becoming involved in software development without a basic education in software engineering. This situation is quite specific to (embedded) software development. In most professions, high educational attainment in subjects closely related to the profession is a prerequisite. This is then usually followed by some years of practical training and experience, before the trainee is moved into a position of responsibility. The only alternative route to most professions involves a long apprenticeship during which both practical and educational achievement is required.

For Philips, the education problem was already signalled in the beginning of the 80's in the reports "A Proposal for a Programme for Software Education" [1] and “A Model for Software Education for Software Developers within Philips” [2]. On the basis of these reports a course programme was defined, and since then courses on computing science and software engineering have been given.

2. Aspects of Engineering

The development process for software (the specification, design, realisation and documentation of software products) can certainly be characterized as a technical activity. But often this development process is claimed to be an engineering activity.

An important aspect of any engineering activity is that there exists some kind of procedure, some kind of production techniques, according to which the activity is performed (the products are realised). A second important aspect is the existence of a thorough basis of fundamental, theoretical knowledge on which the activity is based, with mathematics fulfilling an important supportive role. For electrical engineers, for instance, Maxwell's equations and Kirchhoff's laws belong to the basic knowledge they carry with them. And in her/his design work, every engineer makes use of mathematical models. Even mathematical theories, which may seem at first sight to have little relation to practice, can prove to be very important in practice. The electrical engineer, for example, places great reliance on the theory of complex numbers. A third and very important aspect of engineering is that the inventiveness and creativity of the engineer are embedded in a merge of the science and the relevant production techniques.

In the above the term "engineering *activity*" has been used because we want to make a clear distinction between three different issues with respect to engineering:

- the ***discipline***: the knowledge and skills coming from the merge of science and production techniques (see [3]);

- the ***activity***: how do engineers work in practice, what do they do;
- the ***curriculum***: in the education of engineers attention is paid to other subjects, outside their own discipline.

3. Is It Software *Engineering*?

In considering whether software development could be classified as an engineering activity in our organisation, we have to ask ourselves the following questions:

- Are we in the stage of routine commercial production? Do we have established procedures for the development of software? Do we have skilled craftsmen who are trained in the procedures?
- Is there a scientific basis for software development? Did the software developers receive an engineering education which has provided them with a substantial part of the existing science which they can put to practical use? Do software developers use these knowledge and skills in practice in the development of software?

Much attention is now being paid to the first set of questions. Many software development organisations are not yet in the stage of routine commercial production and are trying to improve their software development process, by using, for instance, the Capability Maturity Model of the Software Engineering Institute.

Some people claim that there are no (or not yet) fundamental theories underlying software and software development, as there are, for instance, for electronics and construction. In other words, they claim that there does not exist a supporting science and that for this reason software development cannot be an engineering activity. But the main requirement in the design and construction of software is the mastering of the complexity of that software. Abstraction and modularisation are needed and for that purpose formal techniques can be used with great advantage. Many formal techniques have been developed or are in the course of development. In the last two decades a body of rather basic science has been crystallised and many advances are still taking place.

The best conclusion for this section can probably be found in [4]:

"My claim is, therefore, that to make software into an engineering discipline we must make sure that the methods we use are scientific, so that they give predictable and verifiable results. Why do we not yet do this? I think there are two reasons:

1. The necessary theory simply does not yet exist for all aspects of development.
2. Many software practitioners do not have the necessary knowledge to use the methods that do exist; more importantly, they do not believe that such methods are useful.

These are not problems that can be solved by education alone: academic researchers must continue to develop the relevant science; practitioners must change their expectations and attitudes from those of craftsmen to those of engineers; and educators must instil both the necessary knowledge and a rigorous engineering approach."

The merge of science and relevant production techniques, the discipline, hardly exists.

4. The Activity

In software development organisations the improvement of the process should go hand-in-hand with an increase in the necessary professionalism of the practitioners, but the scientific foundation needed to underpin true engineering activity is often lacking. People often do not have the appropriate education and a professional engineering attitude is not promoted. There is also considerable opposition to the use of formal techniques. The same applies to every new branch in which practical know-how and tricks of the trade play an important role. It is typical for any craft. Although the situation is improving, we still face a non-professional attitude towards software development in our organisations. Two quotations:

"The 'anyone can be a programmer [software engineer]' attitude is still prevalent in our culture. The only credentials one seems to need are a general familiarity with the syntax of a programming language, a rudimentary knowledge of a text editor, and enough exposure to an operating system to invoke a compiler. [...] If we want software professionals to be considered professionals in every sense of the word, two of the major obstacles we will have to overcome are the inability to think of software as a product, and the idea that little or no skill is required to create and handle software throughout its life cycle." (from [5])

"The key to the professionalisation of software development is a willingness on the part of software developers to accept responsibility for the results and consequences of their work. An important prerequisite for such an assumption of responsibility is a rational, scientific, technical basis." (from [6])

5. The Curriculum

At the universities the interaction between the emerging science and commercial practice proceeds also with great difficulty.

The terminology and language in which the science is expressed is not directed to the software developer. They should be extended to include simpler forms and should be better adapted to the orientation, mentality, and needs of the engineer, as opposed to those of the scientist. The universities educate computer scientists, not software engineers. Computer science shares its interest in formalisms, symbolic structures and their properties with mathematics. On the other hand, software engineering has much in common with constructive sciences such as electrical engineering.

Universities pay hardly any attention to the process

view of software development, almost all attention is paid to properties belonging to the product view.

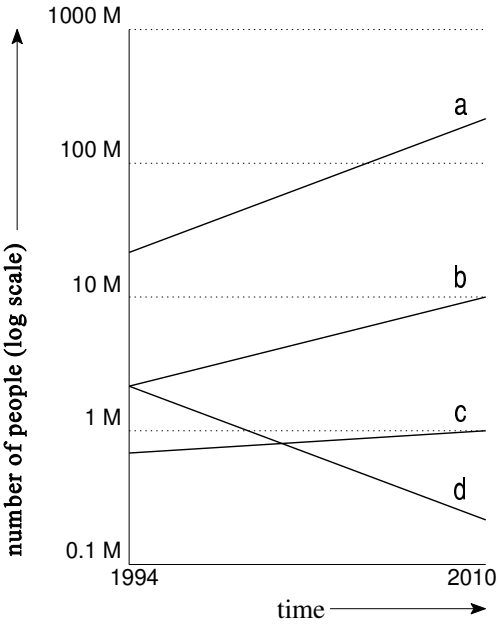
Only a few universities offer a real curriculum for software engineering. Often software engineering is just one single course in a curriculum for computing science. But the software engineering discipline is not a (small) part of computing science. On the contrary, parts of computing science can - after a careful 'translation' - belong to the necessary background for the software engineer. The relation between software engineering, computing science and mathematics can be compared with the relation between electrical engineering, physics and mathematics. In the education of electrical engineers attention is paid to physics and mathematics, but in another way and with another purpose than in the education of physicists and mathematicians. In the same way computing science and mathematics have to play an important role in the curriculum for software engineering, but with another purpose and in a different style than in the curriculum of computing scientists and mathematicians.

According to N. Gibbs ([7]) computing educators need to do the following:"

- Recognize that software engineering in the 1990s is at a state of evolution comparable to that of computing science in the 1960s.
- Gain a better understanding of, and appreciation for, the constraints under which software is developed by practitioners, or risk being ignored by the practitioner community - the ultimate consumer of our educational products - as being irrelevant.
- Recognize that as software engineering evolves, it will be to computing science what mechanical engineering is to theoretical physics. [...]"

6. For whom do we do it?

For Philips, there is a (yet) far too small supply of graduates in software engineering, such that at the intake for the operational level people with an inadequate education have to be engaged. One of the causes is, as we have seen above, that universities have programs for computing science, not for software engineering. A second reason can be found in the number of people we need. In 1994 a workshop with the title "Software 2000 - a View of the Future" was organised in England ([8]). The participants came from Europe, USA and Japan, and were from a wide range of backgrounds (industry, universities, hardware companies, software producers, etc.). They tried to quantify the numbers of people involved in software. Although we should use their estimations (or better: guesses) with precaution, it gives us some indication. They expected the number of software developers for embedded systems to grow from 2M in 1994 to 10M in 2010. Where do all these software developers for embedded systems have to come from?



a: "intermittent" programmers
b: professionals in product companies
c: professionals in software companies
d: IT departments

A 'normal' career in an engineering activity is any initial fragment of the following series: adequate education, design and construction, team and project leading, and decision making. In the case of software the 'adequate education' is an education in software engineering at university or polytechnic level. Evidently such graduates still have to be introduced to their work in an industrial environment, but this holds for all kinds of engineers. For those who will grow into functions of a leading and organisational nature, further training in management techniques will be required, with respect to both general management techniques and special software management techniques.

But many careers in software development are not normal. A typical form of an abnormal software career is the intake of graduates of non-software engineering education or people that shift from other disciplines. A second form of an abnormal software career is the shift of people experienced in a different technical discipline to positions in which they lead software teams and projects. In order to be effective and to create the necessary environment, the software manager must understand the technical aspects of software development. A third form of an abnormal software career is the shift of people at the decision taking levels into the software area or, equivalently, software creeping into their activities.

7. How do we do it

So, Philips needs a lot of software professionals who have had an education in software engineering. Universi-

ties do not provide us with a sufficient number of these people. This means that we must have our own education and training programme.

On the basis of [1] the CSO (Dutch: *C*ursus *S*oftware *O*ntwerp (Course on Software Design)) has been developed. The course takes 35 days. The main purpose of the course is to teach the participants how to apply formal techniques in programming-in-the-small (pre- and post-conditions and formal descriptions of abstract data types). In [1] the following argument for the use of formal techniques is given: "For all but some trivial software systems, the most important characteristic of software is its inherent complexity. Software design and implementation has therefore to do with the mastering of this complexity. Formal techniques and abstraction (as applied in e.g. mathematics) are most essential tools for that".

The ideas of the CSO are used and extended in a second course, the Course on Software Engineering (CSE), which takes 36 days. In the CSE abstraction and formal techniques play an important role too, but now directed to programming-in-the-large. The course consists of four parts: Introduction Software Engineering (ISE), Management Aspects of Software Engineering (MAN), Analysis (ANA) en Design and Implementation (DES). In de last two parts, ANA and DES, formal techniques and object oriented construction play a central role. For didactic reasons we have developed our own model, approach and methods for the software development process: INTOOM (INTegral Object Oriented Methodology).

The constructive aspect plays an important role in our courses. The course programme has further been based on the following premises:

- Formal techniques are important in software engineering. Software engineers must be able to work with formal systems. This implies that they must also be familiar with the mathematical theory and mathematical techniques involved in formal systems (see [1]). A software engineer must be able to handle formal techniques just as an electrical engineer or a civil engineer handles mathematics, and not in the way a mathematician does. For that reason we make a distinction between 'formal' and 'rigorous'. An approach to software development is *formal* when all steps in the development process (from specification to coding) are performed formally, all the techniques and tools that are used are formal (have an exact mathematical meaning), and the use of these techniques and tools is also formally justified. The approach to software development is *rigorous* when the approach uses the formalisms but omits the formal proofs. This may be compared with the electrical engineer who will reverse the sequence of summation and integration in a formula if it suits him; a mathematician will first prove that the relevant function possesses the properties under which the reversal is permissible.

  (Where can we find textbooks for such an approach?)
- In software development a distinction has to be made between the product view and the process view. Some of the activities of the software development process need the results of other activities, they are logically related. Whilst planning, activities are put in a specific chronological order. Often the logical relation between the products, which are the result of the various phases in the development, is equated to the chronological sequence in which the products are realised. We want to disengage the relation between the products from the sequence of realisation. We therefore distinguish in our reference model the product view and the process view. With the product view and the process view both the uncoupling of the logical and chronological order and the uncoupling of the activities and deliverables are made explicit. With this uncoupling we can look separately at the properties of the products and the properties of the processes. Of course there is a relation between the two classes of properties. When we improve the software process this is not only done for improving the productivity, but also because we hope to improve the quality of the resulting products in this way.
- There are three roles in the software development process:
  - *users*: perceiving a need for a software system (application concept), knowing about the application domain but not necessarily about software;
  - *information engineers*: understanding the user's need and specifying the needed system in the commercial requirements specification (in the terminology of the application domain);
  - *software engineers*: realising a software system meeting the software requirements specifications.

  The first document of the software development trajectory is the formulation of user requirements. This is followed by the other documents: the commercial requirements specification, the software specification, the software design description, and finally the construction of the system itself. We can divide the activities which are involved in the development of the system into two groups:

  requirements engineering:

  This comprises the activities needed for capturing the needs of a customer of a software system in an accurate way: the requirements specification (and the software specification). It thus involves the methods, techniques, notations and heuristics necessary for producing a (possibly formal) description of the system (in the terminology of the application domain).

  These are activities of the information engineer.

  design engineering:

  This comprises the activities needed for realising an adequate software system to satisfy the needs of a user. It can include the software specification. It thus involves the methods, techniques, notations and heuristics for correctly implementing the system, starting from (or with) the software specification.

These are activities of the software engineer.
(Where in textbooks can we find this kind of explicit choices?)

8. The Future

The courses have been given now for more than ten years. We do not intend to give these courses forever. We are now in the process of defining again our requirements with respect to the knowledge of the software developers and the courses that can fulfil our wishes. We do like the core courses in the Software Engineering curriculum of Carnegie Mellon University [9] and it wouldn't be a surprise if the specifications of our courses would have a lot in common with the descriptions of these core courses. With the specifications we will go to universities and ask them to define and teach the courses. Will they be able to do so?

References

[1] R.H. Bourgonjon, F.E.J. Kruseman Aretz, P. Medema, G. van der Munnik, B.L.A. Waumans
A Proposal for a Programme for Software Education
Philips, Eindhoven, 1981

[2] R.H. Bourgonjon, F.E.J. Kruseman Aretz, P. Medema, G. van der Munnik, J.A.A.M. van Oers, B.L.A. Waumans
A Model for Software Education for Software Developers within Philips
Philips, Eindhoven, 1984

[3] M.Shaw
Prospects for an Engineering Discipline of Software
IEEE Software, November 1990

[4] A. Hall
Is Software Engineering?
In: C. Sledge (ed.)
Software Engineering Education
(SEI Conference 1992)
Springer-Verlag, Berlin, 1992 (LNCS 640)

[5] E.Berard
Towards a Software Profession
Structured Language World, 1986

[6] R.L.Baber
Portrait of a (Software) Engineer
Journal of Systems Software, 1991

[7] N.E.Gibbs
Software engineering and computer science: the impending split?
Education & Computing, 1991

[8] B.Randell, G.Ringland, B.Wulf
Software 2000: a View of the Future
Commission of the European Communities, Brussels, 1994

[9] D. Garlan, A. Brown, D. Jackson, J. Tomayko, J. Wing
The CMU Master of Software Engineering Core Curriculum
Carnegie Mellon University (CMU-CS-93-180), Pittsburgh, 1993