The Teacher: "Concepts!" The Student: "Tools!" — On the Number and Importance of Concepts, Methods, and Tools to be Taught in Software Engineering Education

Martin Glinz
Institut für Informatik
University of Zurich, Switzerland
glinz@ifi.unizh.ch

Abstract

The paper discusses the number and importance of concepts, methods, and tools that should be taught in software engineering education. It is shown that emphasis is quite different from a teacher's and from a student's viewpoint. As a synthesis of both viewpoints, I propose an approach that separates concepts and methods from tools in the curriculum and treats both with proper emphasis. The software engineering curriculum of the Department of Computer Science of the University of Zurich which follows these ideas is sketched.

1. Introduction

When looking at the preferences for concepts, methods, and tools in software engineering, one finds considerable differences between the viewpoints of teachers and students. Most teachers lay emphasis on concepts and methods due to their generality and scientific importance. Students typically prefer tools because they are practical and provide hands-on benefit.

In this paper, I use the terms concept, method, and tool as follows. A *concept* is a basic, abstract idea for representing or solving a problem. A *method* is a strategy of how to apply a concept in practice using a given notation. A *tool* supports the application of a method and its associated notation (Glinz [5], Ludewig [6]).

As both teachers and students have good arguments for their viewpoints, a good software engineering curriculum should find a suitable synthesis between them. In this paper I sketch such a synthesis. It is characterized by four principles: "Many concepts – some methods – few tools", "Separate concepts from tools", "Introduce concepts early", and "Make tools useful for students".

The paper is organized as follows. In the next two sections I describe the teacher's and the student's viewpoints, respectively. Then I present my view of a synthesis. In section five I give an overview of the software engineering curriculum at the University of Zurich, which follows these ideas.

2. The teacher's viewpoint

From a teacher's viewpoint, the selection of topics in software engineering education and the emphasis of teaching these topics should be strongly correlated with the generality and the lifetime of the topics.

When looking at concepts, methods, and tools, we find a decreasing order both for generality and lifetime. Concepts form the scientific foundation both for methods and tools. So, by their very nature, they are general and long-lived. Methods typically are instances of one or more concepts and make them practically applicable. Tools improve productivity when applying a method. Therefore, methods are less general than concepts but more general than tools. When looking at the history of software engineering, one can also see that methods typically live shorter than concepts, but longer than tools. For example, the concept of layered, dataflow-based system models originates in the early seventies. Various methods have been devised based on this concept, e.g. Structured Analysis (DeMarco [2]), Essential Structured Analysis (McMenamin and Palmer [7]), SADT (Ross [9]), and SSADM (Ashworth and Goodland [1]). A lot of tools have been developed for these methods. Many of them are already part of history and are no longer used.

From a purely teaching standpoint, use of tools can even be considered harmful: Learning how to use a tool takes a considerable amount of time. Students concentrate on mastering the tool (and on playing around with it) instead of solving the problem at hand. That means, tool use emphasizes syntactic issues instead of semantic ones.

3. The student's viewpoint

For most students, tools are the real stuff. It is the tool that helps her or him writing, compiling, and debugging programs. It is the tool that makes application frameworks accessible and usable. It is the tool that saves time. It is the tool that is fun to play with. So, learning by tool usage is highly motivating for students.

Methods are frequently considered to be some kind of abstract guide to how to use a tool. Learning them is motivated by the experience when using tools.

Concepts, however, tend to be regarded as theoretical issues by the students. Left to themselves, most students find it difficult to understand why they should learn them. There is nearly no intrinsic motivation to study concepts in software engineering. Showing the value of concepts by demonstrating their application and impact in practice is among the most challenging tasks in software engineering education.

4. Synthesis of viewpoints for a software engineering curriculum

4.1 Many concepts – some methods – few tools

Due to the short half-life of knowledge in computer science, the curriculum must concentrate on the fundamental and long-lived topics – concepts and methods in our context. Building on this foundation, the teacher must show the students how to embody and apply these concepts and methods in practice using selective examples and – where useful – tools.

The focus should lie on a broad teaching of concepts, because methods are less general and far more numerous than concepts (they typically are instances of concepts). On the other hand, methods are less abstract and therefore easier to understand and to apply. Therefore, some methods should be taught to show the concepts in a form that is practically applicable.

A few tools should be used in laboratory exercises with two purposes: (a) to show the principal possibilities of tool support and (b) to teach the students how to quickly acquire a basic knowledge of a previously unknown tool.

Concepts can neither be explained nor exercised without using a notation. This notation can either be generic or it can be associated with a particular method. For example, Ghezzi, et al. [4] use a generic design notation in their textbook. Pressman [8], on the other hand, presents concrete notations with their associated methods, e.g. Structured Analysis or JSD.

Generic notations have the advantage of primarily conveying the concepts. However, when using generic notations only, the students do not learn any notation being used in practice. Moreover, generic notations may be perceived as being artificial and theoretical by the students, thus lowering their motivation to learn and use them. Therefore, a fair mix between generic and real notations seems to be optimal in a software engineering curriculum.

4.2 Separate concepts from tools

Concepts should be introduced without making use of tools. Exercises using paper and pencil help to focus on semantics instead of being stuck with syntax and tool handling.

When I taught software requirements engineering courses in industry some years ago, I encouraged the participants to do their exercises either with paper and pencil or with a state-of-the-art CASE-tool. The experience was that the participants using paper and pencil came up with better specifications in a shorter time. Even participants who had already some experience in using the tool were outperformed.

4.3 Introduce concepts early

Software engineering concepts are part of the basic knowledge that should be taught early in the curriculum – like mathematics, computability, or algorithmic complexity. In this way, courses introducing methods or labs using tools can build upon this knowledge.

4.4 Make tools useful for students

On the other hand, we have to recognize the importance of tools for students. Therefore, a few carefully selected tools should be introduced that support the work of the students and improve their personal productivity. Typical candidates are programming environments, project management tools, or tools that build systems based on frameworks.

5. The Software engineering curriculum at the Department of Computer Science of the University of Zurich

In our department we are currently introducing a new curriculum (Dept. of CS [3]). Like most universities in German speaking countries, our curriculum consists of four semesters (two years) of basic studies followed by at least four semesters of advanced studies. The students graduate with a diploma degree (Dipl. Inform.).

Table 1 gives an overview of the software engineering courses in our curriculum together with their employment of concepts, methods, and tools.

In the advanced studies, software engineering is one of eight fields from which the students have to choose three or four for their studies as options. Additionally, there is a set of compulsory courses (the core field, as we call it), which comprises a software engineering course, too.

The software engineering component of our curriculum follows the ideas sketched in the previous chapter to a considerable extent.

We do introduce concepts early. In the course on modeling for example, we introduce the notion of mod-

Table 1. Software engineering courses at the Department of Computer Science of the University of Zurich

Course title	Semester	Lessons	Usage of concepts, methods, and tools
Basic studies			
Programming+	1 and 2	39	Concepts plus a programming environment as a personal tool
Models in computer science+	2	18	Concepts only
Introduction to software engineering+	3	26	Concepts with some methods
Software engineering lab+	4	52	Practical exercises how to apply basic techniques of software engineering using methods and few selected tools.
Advanced studies			
Software quality management+	6	26	Concepts with some methods
Individual software project+	≥6	3 months full time in industry	Students shall apply concepts and methods. Tools are used when available in the company where work is carried out
Advanced topics in software engineering *	≥5	26	Advanced concepts and methods
Requirements engineering*†	≥5	26	Concepts with some methods
Software architecture and design*†	≥5	26	Concepts with some methods
Advanced topics in programming*†	≥5	39	Concepts with some methods
Special courses in software engineering, for example: Prototyping, Software re-engineering, Object-oriented systems development, Logic programming, Functional programming, etc.	≥5	26 each	Concepts with some methods in most courses
Software project management lab**	≥5	26	Methods and a project management tool
IS Development lab**	≥5	26	Methods and a CASE-tool
Logic programming lab**	≥5	26	Methods and a programming environment

\+ Compulsory for all CS students
* For students choosing software engineering as one of their options
** Optional course
† Students may replace two of these courses by two from the set of special courses in software engineering

els and abstractions as well as the fundamental concepts of modeling data/objects, structure, behavior, and functionality.

We make tools useful for students by using a programming environment and by experimenting with a few tools in the software engineering lab.

We separate concepts from tools by restricting tool use to programming and to some exercises in the lab courses.

Finally, Table 1 shows that we follow the principle of "Many concepts – some methods – few tools".

6. Conclusions

I have presented some thoughts on the role of concepts, methods, and tools in software engineering education. The principal idea is that "Many concepts – some methods – few tools" represents an optimal mix for a software engineering curriculum. Concentration on concepts emphasizes the topics which are really important for the students. Methods and associated notations help to understand and apply the concepts. Using a few tools demonstrates the capabilities and the limitations of computer-aided software engineering. Furthermore, "real-life" methods and tools increase the motivation of the students.

The curriculum of our department that I have presented as an example is currently in the phase of introduction. Therefore, I cannot report any experience yet. However, from my earlier experience in software engineering education in industry, I am confident that we are on the right path.

References

[1] Ashworth, C. and M. Goodland (1990). *SSADM: A Practical Approach*. McGraw-Hill, London.

[2] DeMarco, T. (1978). *Structured Analysis and System Specification*. Yourdon Press, New York.

[3] Dept. of CS (1995). *Wegleitung für das Studium der Wirtschaftsinformatik an der Universität Zürich* [Guide for studying business-oriented computer science at the University of Zurich (in German)]. Version 1.3. Department of Computer Science, University of Zurich.

[4] Ghezzi, C., M. Jazayeri and D. Mandrioli (1991). *Fundamentals of Software Engineering*. Prentice-Hall, Englewood Cliffs, N.J.

[5] Glinz, M. (1990). Warte nicht auf bessere Zeiten: Methoden und Werkzeuge in der Softwareentwicklung. [Do Not Wait for Better Times: Methods and Tools for Software Development (in German)]. *Technische Rundschau* 35/90. 70-75.

[6] Ludewig, J. (1985). A Note on Abstraction in Software Descriptions. In D. Teichroew and G. David (eds.): *System Description Methodologies*. Elsevier Science (North-Holland), Amsterdam. 535-540.

[7] McMenamin, S.M. and J.F. Palmer (1984). *Essential Systems Analysis*. Yourdon Press, New York.

[8] Pressman, R.S. (1992). *Software Engineering - A Practitioner's Approach*, 3rd edition. McGraw-Hill, New York.

[9] Ross, D. (1977). Structured Analysis (SA): A Language for Communicating Ideas. *IEEE Transactions on Software Engineering* SE-3(1). 16-34.