Teaching Software Modeling in Computing Curricula

Jürgen Börstler
Blekinge Institute of Technology
Karlskrona, Sweden
jubo@acm.org

Ludwik Kuzniarz
Blekinge Institute of Technology
Karlskrona, Sweden
lku@bth.se

Carl Alphonce
University at Buffalo
Buffalo, USA
alphonce@buffalo.edu

William B. Sanders
University of Hartford
West Hartford, USA
wsanders@hartford.edu

Michal Smialek
Warsaw University of Technology
Warsaw, Poland
smialek@iem.pw.edu.pl

ABSTRACT

Modeling is a key skill in software development. The ability to develop, manipulate and understand models for software is therefore an important learning objective in many CS/SE courses. In this working group, we investigated how and when (software) modeling is taught to help us better understand the key issues in teaching (software) modeling. Several shortcomings were found in common curricula, both in their understanding of the term "modeling" and in how they address its teaching. This WG report summarizes the findings and formulates recommendations on the inclusion of software modeling courses in future CS/SE curricula.

Categories and Subject Descriptors

D.2.10 [**Programming Techniques**]: Design—*Methodologies,Representation*; K.3.2 [**Computers and Education**]: Computer and Information Science Education—*Computer science education*

General Terms

Design, Human Factors

1. INTRODUCTION

Modeling is essential in any constructive human activity. Models are used both for understanding existing artifacts, concepts, or phenomena, as well as for creating new ones. Modeling is widely used in science and engineering where, e.g., mathematical models are used for expressing laws that rule the physical phenomena. In general, models facilitate solving complex engineering problems through focusing on the aspects that are most relevant in a specific context and abstracting from the details that are (currently) irrelevant. This ability of modeling to cope with complexity is claimed to be of significant importance to the field of Computing [19, 26]. For many years already, Software Engineering has worked out specific sets of modeling techniques and notations that distinguishes software models from the models developed in other engineering disciplines[1]. In many software development methodologies, models, such as conceptual models, domain models, design models, workflows, or flowcharts, are primary artifacts. It can be noted though, that the meaning of the notion "model" is still not agreed upon in software engineering. The question "what is a model?" is still open and seems not to be completely resolved [25, 20]. Despite this, the abilities to understand, develop and properly use models are important learning objectives for anyone involved in the software development processes.

This importance was recognized through the inclusion of specific software modeling topics in various courses within the CS, SE and IT curricula. Following these first attempts, modeling became more widely present in such curricula, and finally included in several curricular recommendations. The ACM/IEEE curriculum guidelines for undergraduate degree programs in Software Engineering (SE2004)[2] emphasizes that the development of software "places a greater emphasis on abstraction, modeling, information organization and representation, and the management of change" [16]. The guidelines for Graduate degree programs in Software Engineering (GSwE2009) demand that "students must be able to recognize the importance of abstraction and modeling for software architecture, design, and specification" [27]. The words *model* and *modeling* are used numerous times in the GSwE2009 Core Body of Knowledge.

Despite modeling's mention in curricular recommendations, there seems to be no clear, agreed upon definition of what software modeling is. It seems to follow the above mentioned uncertainty on the term "model" in software engineering. This is a crucial problem, because in order to define clear learning goals, we need to define the scope of teaching. Considering this, the goal of this work was to gain a better understanding of what is taught in the context of software modeling and when it is taught. We collected data from several different sources and, based on this data, an attempt was made to review the current state-of-the-art and state-of-practice in teaching of software modeling. This allowed us to point out some problems with the current status and formulate recommendations for the future.

2. RELATED WORK

There is a large body of research on modeling in the context of mathematics, science, and engineering teaching in general

[1]In this present work we concentrate on software modeling as contrasted with general modeling in science/engineering, expressed through, e.g., mathematical models meant for simulation,

[2]Curriculum recommendations are listed in Appendix B.

(e.g.,[14, 17]). Regarding modeling in CS/SE/IT education, the literature mainly focuses on specific aspects of modeling, like object-oriented design or modeling languages and tools. There is very little work on the overall role of modeling in CS/SE/IT education. The reasons for this can be sought in general negligence of modeling in Software Engineering curricula. Such claims are formulated by Cowling [10, 11] who has analyzed SE curriculum guidelines and compared with one of the established SE undergraduate programs. Cowling sees modeling as a topic that should be recurring throughout the whole SE curriculum. Though, unlike for the current work, Cowling's conclusion was based on a limited set of input data.

In contrast to Cowling's conclusion, software modeling is often considered as a topic to be taught mostly to graduate students. However, there is some evidence that modeling "tools" can successfully be introduced very early to support students' understanding of object-oriented concepts [7, 8, 12, 29]. In fact, the German secondary school curriculum recommendations for computer science includes significant coverage of software modeling starting at an early age (see section 4.1 for more discussion).

A number of problems and issues related to modeling education were presented and discussed at the Educators' Symposia of MoDELS conferences[3]. The perspectives of these works were usually specific and local, though, focusing mostly on particular courses in particular contexts, rather than on the role and the place of modeling in CS/SE/IT curricula in general [2, 5, 13]. A broader view was taken by Kuzniarz and Staron [22] who proposed a number of best practices for teaching modeling/UML and particularly emphasize the role of consistency between different models or views. Also Vallino [30] takes a broader view and considers software development as fundamentally an engineering endeavor. He presents a Software Engineering program with a strong focus on modeling that "strives to instill a culture of engineering practice by exposing students to both formal and informal modeling of software systems throughout the entire curriculum". Roberts [28] studied the link between abstraction skills and success in Computer Science. She found a positive correlation between abstract thinking skills and success in an object-oriented modeling course. Finally, Bezivin et al. [4] highlight some key questions that should be asked when discussing teaching modeling. Their *Why*, *What*, *How*, and *When* perspectives on the teaching of modeling were extended into a simple taxonomy for the categorization of issues in (research on) teaching modeling [21]. The latter works go into the same direction as the work presented here, but are not based on empirical data.

Cabot and Tisi [9] describe a full-year post-graduate program in Model-Driven Engineering focusing on "the rigorous use of (software) models as the main artifacts in all software engineering activities." The contents of the curriculum is quite advanced and although entry requirements are high (Master degree or equivalent), students had difficulties understanding the relationships between all the concepts.

To our best knowledge, no comprehensive analysis of curricula from the software modeling perspective has been performed. For instance, the ACM/IEEE curriculum guidelines for Software Engineering programs provide valuable input on curriculum design, in particular the perceived relevance or importance of various topics [16, 24, 27]. However, it does not analyze software modeling. Several papers investigate what is or should be taught in CS/SE/IT curricula, most notably Lethbridge's survey [23] on the relevance of topics for practitioners and some follow-on studies [6, 18, 31]. Lethbridge concludes that there is a mismatch between what is taught in computing programs and what graduates of those programs reported as having been important in their subsequent working lives. Computing programs included computer science, computer engineering, software engineering, information systems, other science or engineering majors, or other disciplines altogether. Again, no particular analysis of software modeling was made. Recently, Zendler et al. [32, 33, 34] did several cluster analyses to investigate central concepts for computer science education. The results of these works will be discussed in some more detail in Section 4.4.1.

3. RESEARCH QUESTIONS AND DATA COLLECTION

The main goal of this work was to map the current state-of-the-art and state-of-practice in the teaching of software modeling. To reach this goal, an attempt was made to answer several research questions.

1. **Curriculum recommendations.** What do common curriculum recommendations and guidelines propose?
2. **State-of-practice.** How is the teaching of modeling approached in typical CS/SE curricula?
3. **Definition of software modeling.** What would a commonly accepted definition of "software modeling" look like?
4. **Educator perceptions.** What do experienced educators think about the role and importance of the teaching of software modeling?

Answering these questions can help us to better understand the overall role and importance of software modeling and improve the way it is taught in computing curricula. To base recommendations on an as complete as possible picture of the area, we collected data from several different sources.

1. Curriculum recommendations and guidelines
2. Descriptions of actual courses that teach modeling (not necessarily primarily, but to a non-negligible extent)
3. Perceptions of experienced teachers
4. Relevant literature

[3]The ACM/IEEE International Conference on Model Driven Engineering Languages & Systems "is the premier conference series for model-based software and systems engineering which since 1998 has been covering all aspects of modeling, from languages and methods to tools and applications".

It can be noted that the data sources reflect the research questions. However, some of the sources were used to answer several questions. For instance, perceptions of teachers could be used to extract also individual definitions of modeling. In detail, the sources can be divided into existing documents and the results of a survey prepared within this work. Both kinds of sources were analyzed using predetermined guidelines.

In total eight *curriculum recommendation and guideline* documents were included in our analysis. These recommendations were reviewed using a predefined format (see Appendix C for details). Since some recommendations were not available in English, not all of the recommendations could be reviewed independently by several authors. A complete list of the reviewed guidelines can be found in Appendix B.

For *descriptions of actual courses*, attempts were made to find course syllabi through university web pages. Since this information was more difficult to find than expected, the example syllabi were also requested directly from colleagues from other universities and through the SIGCSE mailing list. In total, 18 course descriptions were collected. It has to be noted, though, that the descriptions are on quite different levels of detail.

For the *perceptions of teachers*, a questionnaire was sent to teachers, known to be very experienced in teaching modeling. In the survey, questions were asked about their own definition of modeling, the courses taught at their universities and about specific topics they find important, unnecessary or lacking in these courses. The full questionnaire can be found in Appendix A. The survey was sent to 34 teachers, of which 22 responded, but only 21 answered all of the questions. For most of the questions, we therefore only have 21 data points. Respondents come from 11 different countries (USA: 7 respondents, Nordic countries: 5, Western Europe: 4, Eastern Europe: 6). Most respondents are on professor level and have substantial teaching experience. Most of the respondents are also researchers. Six respondents also report substantial industry experience. Two of them are textbook authors.

Regarding *relevant literature*, specific conference proceedings and journals were searched. Backward and forward snowballing was also performed.

4. RESULTS

In the following paragraphs we summarize the results of our analysis. Subsection 4.1 elaborates on the treatment of teaching modeling in common curriculum guidelines (research question 1). Subsection 4.2 presents a short contents analysis of the collected course descriptions (research question 2). Sections 4.3 and 4.4 present an analysis of definitions of software modeling (research question 3) and perceptions of importance of certain modeling topics (research question 4), respectively.

4.1 Modeling in CS/SE/IT curriculum recommendations

The treatment of software modeling varies widely in the reviewed curriculum recommendations or guidelines. The most thorough treatment of modeling could be found in the Software Engineering (SE) curriculum recommendations (SE2004 and GSwE2009) and the German guidelines for teaching computer science in secondary school (GI). For the SE curriculum guidelines this was not surprising, since modeling is considered a core knowledge area or unit of SE.

> Modeling and analysis can be considered core concepts in any engineering discipline, because they are essential to documenting and evaluating design decisions and alternatives. (SE2004, p. 25)

The guidelines for secondary schools in Germany had the most thorough treatment of modeling of all guidelines. This is rather surprising, since these recommendations are targeted for students of age groups 10–12 and 13–15. In these guidelines modeling cuts through all topics and is seen as a basic and general tool for problem solving, understanding, and communication.

> The process of modeling is not only learning content but also a consistent method of computer science education. The students learn methods and techniques of modeling that must be developed gradually, based on the actual context. They will apply their knowledge and critically investigate the results of the modeling. . . . Different views on the problems lead to different ways of modeling. . . . (GI, p. 45–46, translated by first author)

All the guidelines, apart from the GI, seem to perceive modeling as an advanced knowledge area within software development, which requires a number of knowledge units as prerequisites, as well as maturity. It is therefore taught during later years of education. In the SE2004 guidelines, this is formulated as follows:

> . . . some core units . . . clearly must be covered only after students have developed significant background in the field. For example, topics in such areas as project management, requirements elicitation, and abstract high-level modeling may require knowledge and sophistication that lower-division students do not possess. (SE2004, p. 18)

Is is rather surprising to find such clearly opposite views. On one hand, modeling is considered a basic and general skill that can and should be taught as a recurring topic starting very early. On the other hand, it is seen as an advanced topic that can only be taught to advanced and mature students.

A summary of the main "features" of all the curriculum guidelines can be found in Figure 1. For each of the eight documents, a brief summary of its contents and treatment of modeling is made. The documents generally target various Computer Science (CS), Software Engineering (SE) and Information systems (IS) courses taught as undergraduate or graduate. Only one of the documents treats pre-university education. They vary in length significantly (from 10 to

						Modeling			
Curriculum guideline	Origin	Target group	Length	Actual syllabi	Mapped to KA/KU	Definition of ...	Importance paid to ...	Teaching Guidelines	Comments
CS2013	ACM/IEEE[2]	CS under-graduates	172+[1]	Y[1]	Y[1]	N	Little	N	Software modeling is discussed as an activity subsumed by software design.
GI recom-mendations	DE	CS at school level	72	N	N	Implicit	Very high	Recurring topic, examples	Although it is never said explicitly that modeling and abstraction are very important, this shines through the way knowledge, skill, and learning goals are defined.
Graduate SE	ACM/IEEE[2]	SE graduates	124	N	N	Implicit	High	Vague, some examples of modeling	It seems that the authors have been very careful in avoiding words like specifying or programming something; they rather talk about creating solution descriptions (as one purpose of development [p 74]).
IFIP	UNESCO	CS/IS/SE higher education	147	Y	Y	N	Some	N	Information modeling is one of 12 "core curriculum themes".
LACS	US	CS college (LA)	35	Y	N	N	Little	N	LACS2007 is essentially silent on modeling.
MSIS	US	IS graduates	76	Y	N	Explicit, but vague and narrow	Some	N	There is no clear and consistent understanding of modeling. The only elaborated description is limited to conceptual modeling. There are no clear references to modeling in other areas of system development.
PLCS	PL	CS/CE under-graduates and graduates	10	N	N	Implicit	Medium	N	Modeling is included explicitly, but treated quite vaguely. Mainly associated with information systems analysis.
SE2004	ACM/IEEE[2]	SE under-graduates	135	Y	Y	Implicit	High	Recurring topic	Heavy use of UML as a topic. "Modeling and analysis can be considered core concepts in any engineering discipline, because they are essential to documenting and evaluating design decisions and alternatives." [SE2004, p 25] "... abstract high-level modeling may require knowledge and sophistication that lower-division students do not possess." [SE2004, p 18]

Y=Yes, N=No, KA/KU=Knowledge Area/Knowledge Unit

[1]The Strawman version of the document did not contain actual syllabi and mappings to KA/KU. These will be included in the final version.

[2]Claims to be international, but focuses on the North American, Australian, and UK educational systems.

Figure 1: Summary of reviewed curriculum recommendations.

172 pages). Some of them present actual course structures and contents, others just recommend the overall distribution of teaching materials. Moreover, some of the recommendations define Knowledge Areas or Knowledge Units that are then mapped to the contents of specific course structures. As Figure 1 shows, the individual documents also vary significantly in their treatment of software modeling. Practically none of the recommendations presents a clear and explicit definition of software modeling, although this term is used in several places. General CS courses seem to pay little attention to modeling. Only two documents (SE2004 and GI) treat software modeling as a recurring topic taught throughout the whole program. The last column in the figure contains a summary comment taken from a review or from the guideline/recommendation itself.

4.2 Common modeling courses

The survey respondents were asked to describe the courses taught at their universities that have at least some elements of modeling included. This allowed us to analyze 21 approaches to including modeling into CS/SE/IT curricula. Altogether, 81 courses were reported. This makes an average of almost 4 courses per curriculum covering both undergraduate and graduate courses. Of all these courses, 12 were dedicated specifically to software modeling. Some elements of modeling (e.g., the UML notation) were included in 9 reported general courses on the foundations of software engineering. This was supplemented by 7 courses on software architecture (basic and advanced) and 7 courses on software analysis and design. Such courses are obvious candidates for containing significant amounts of modeling. Modeling is also sometimes present in programming (object-oriented and general). There were 5 such courses reported. Database courses were also reported (5 in total) as including elements of modeling, although this figure can be argued as being underestimated. Many database courses use some form of conceptual or relational modeling. In addition, respondents reported several other courses (2 each) that included some elements of modeling: requirements engineering, software quality, discrete math / data structures, artificial intelligence (AI) / knowledge representation (semantic web) (KR), business process management. Finally, some specialized and general knowledge courses included modeling: software product lines, systems development, computer graphics, emerging technologies, CASE tools, and distributed objects. Table 1 shows a summary of this data.

Table 1: Courses with modeling content

course topic/area	# courses
software modeling	12
sofware engineering	9
software architecture	7
software analysis and/or design	7
programming	5
databases	5
requirements engineering	2
software quality	2
discrete math / data structures	2
AI / KR	2
business process management	2
specialized courses	6

It has to be stressed that this data was provided by experienced software modeling teachers and might therefore not be representative for CS/SE/IT curricula in general. It can be contrasted with the results of the analysis of the curricula recommendations in the previous subsection. Still, the above summary of courses shows a very large diversity and often incoherence in treating modeling throughout the course of studies. What is more, modeling – similarly to what can be found in the curriculum recommendations – is seen as an advanced topic taught later in the course of studies. If a separate course on modeling exists, it is usually taught after the student has already established certain understanding on how software development should be performed.

4.3 Definitions of Software Modeling

Together with the course descriptions, the survey respondents provided 21 definitions of software modeling. It has to be emphasized that the respondents were not asked about formal definitions, but about their own informal understanding of modeling. However, we believe that they give a good coverage of how software modeling is understood in the context of teaching. This constitutes a good basis for performing an analysis of the spectrum of approaches to define the knowledge area of software modeling. To formalize this analysis, four questions were formulated related to different aspects of the definitions.

Goal orientation. How strong is the definition's focus on the purpose of modeling ("Why" aspect)?

Model as a thing. How strong is the definition's focus on defining what is a software model ("What aspect")?

Modeling as a tool or process. How strong is the definition's focus on how to perform software modeling ("How aspect")?

Generality. How general is the definition in covering various types of software modeling ("Where aspect")?

It can be noted that these questions go along the work by Bezivin et al. [4] mentioned in Section 2. However, the "when" question was substituted with the "where" question. This was done because the definitions were not supposed to define when modeling should be taught. Instead, the generality of the definition indicates how modeling is perceived in relation to various other aspects of software development.

For each of the questions, scores from 0 to 5 were given subjectively by four of the authors. The score of 5 indicated that the definition ideally stresses the given aspect of modeling. The score of 0 indicated that the definition does not cover the given aspect. These scores were used to determine "good" and "bad" definitions through simple summing of the four scores. It was assumed that a "good" definition should stress all 4 dimensions.

The three highest scoring definitions earned from 56 to 59 points in total from possible 80, and are as follows:

- "Modeling is a core activity of software engineering by which an abstract representation of the problem or its solution is developed (which are usually different models). The nature of the model (including its underlying language) depends on the goal of the modeling (the same reality leads to different models, depending on purpose)."
- "Modeling is any activity involving abstraction, where a consistent set of rules is applied to simplify a complex situation for greater understanding. Modeling: the task of building system representations, generally using a graphical modeling language, in order to improve the software development process. Besides automatic code generation, models are also important to reason about systems, to ease communication among stakeholders, and to document design decisions."
- "Modeling is the process of designing abstract descriptions of software with the purpose of analyzing aspects/properties, designing solutions or communicating information about these to other parties."

The four lowest scoring definitions earned from 23 to 29 points, and are as follows:

- "Modeling in the context of software development is one of the most important and most difficult stages in software development. It's got a decisive influence on the quality of the product and its maintenance."
- "Modeling is a method for describing problems and solutions at a higher abstraction level, than programming. In general the boundary between programming and modeling is quite blurred."
- "Representing the logical and physical structure of code and providing an abstraction of its behavior."
- "Designing classes and collaborations between classes to most effectively model the concepts in the business domain."

It has to be noted that the above definitions were marked as "good" and "bad" considering the analysis criteria. The respondents did not know the criteria and thus were not influenced to formulate a "good" definition.[4] It can thus be argued that this resulted in better overview of how modeling is perceived by the educators in their teaching practice. The definitions vary significantly. Some of them describe modeling broadly, and emphasize the purpose of modeling that goes beyond pure software engineering goals. Some of the definitions limit the scope of modeling to, e.g., low level design (close to code), or business modeling. Some of the definitions acknowledge variety in modeling approaches that depend on the purpose of modeling.

To complement the analysis, we performed a word frequency analysis of the 21 informal definitions. Additionally, this was compared with the word frequencies of 9 definitions from standards and glossaries[5]. Table 2 summarizes the differences and similarities for the most frequent words.

[4]The definitions were first de-personalized and were treated as anonymous. The scoring was made only for the purpose of this analysis and no scoring of the respondents was made.

[5]Examples of standards and glossaries consulted are: IEEE Std 1012-1986, U.S. FDA Glossary of Computer Sys-

Table 2: Word frequencies in definitions of model/modeling.

Word	Survey (21)	Glossaries (9)
abstraction	12	2
communication	8	0
design	8	5
program/code	8	1
understanding	7	0
representation	6	8
problem	3	1
analysis	2	4
reality	1	7[6]
relationship	0	2

Interestingly, the IEEE Standard Glossary of SE Terminology does not define the terms "model" or "modeling". However, "model" is used frequently in definitions of other terms, e.g., waterfall model or simulation. Similar observations can be made for other glossaries; "model" is used as a commonly known term that does not need an explanation. For these glossary definitions that contain software modeling, it is characteristic that word frequencies differ to large extent as compared to the wording used by practitioners in the survey definitions.

The above analysis shows that software modeling should be understood as a broad activity that crosses through most of the activities in the software engineering process: analysis, design, programming. What is more, it encompasses activities that go far beyond just software engineering. Modeling is considered as a tool for realizing abstractions and facilitating communication, or more generally – representing reality so that other people or machines could precisely understand some aspect.

4.4 Importance of specific modeling topics

The fourth part of the study consisted in analyzing topics that are believed to be important among practitioners of the field, and also reported in literature. It has to be noted that there seems to be no systematic analysis dedicated to specific modeling topics available. Furthermore, there was no systematic data on this available, and it had to be collected through the already mentioned survey. The respondents were asked to define specific software modeling topics that are most important, least important and lacking in the modeling courses and in the computing curricula in general.

The generally shared view of teaching modeling focuses on moving students into a realm of abstract thinking for problem-solving related to computer programming. However, in measuring what faculty considered important, a good deal of variation could be found. Conversely, when looking at what faculty believed to be unnecessary, relatively high consensus occurs. What faculty considered lacking was more varied, but often pointed to unavailable and undeveloped teaching tools. The following subsections extend this brief summary presenting more details.

4.4.1 Topics identified in the literature

As already mentioned, we could not find any empirical studies on software modeling in computing curricula. However, several general studies on computing curricula contain topics that are well established as closely related or being part of the software modeling knowledge. Prominently, the Lethbridge survey [23], conducted in 1998, ranked 75 topics extracted from university curricula and SWEBOK (Software Engineering Body of Knowledge [1]). Alumni from several institutions, from various countries, were represented in the survey. Approximately half had graduated in the 1990's, one quarter in the 1980's (8–18 years prior to the survey), and the remaining had graduated between 1950 and 1980[7]. After ranking, the following modeling-related topics[8] were found to be in the top 25: software design and patterns (rank 3), software architecture (rank 4), requirements gathering/analysis (rank 5), and analysis and design methods (rank 9). No modeling topics were found among the low 25 topics. Furthermore, it can be seen that the modeling topics had a fair amount of knowledge that needed to be learned after education. [Figures 2a,2b, page 49]

More recently, two other studies [18] and [31] produced a ranked list of topics within computing curricula. Both were follow-on studies, focusing on computing programs in the UK and Brazil respectively. The Kitchenham et al. study [18], was conducted in 2002 and focused on graduates from 1995 and 1998. It modified Lethbridge's survey instrument and the manner in which participants were selected, so the results are not directly comparable to Lethbridge's. Similarly, the von Wangenheim and da Silva [31] results are not directly comparable to either of the earlier surveys. It was conducted in 2008, targeting students who graduated between 1998 and 2005.

Of the topics in the Kitchenham et al. study, requirements gathering/analysis (rank 5), software architecture (rank 8), analysis and design methods (rank 10), software design practices (rank 12), and software design patterns (rank 17) are modeling-related. All of these modeling topics were also among the top ranked topics for "usefulness of extra training" [Table 3, page 330]. In the von Wangenheim and da Silva study, requirements development (rank 12), software architecture (rank 14) and software design and patterns (rank 15) are the only modeling-related topics, ranked by importance [Table 6, page 19].

Despite that the studies used somewhat different methodologies, they resulted in lists of recurring modeling-related topics

tems Software Development Terminology, Computing at School Working Group (`http://www.computingatschool.org.uk`), `http://informatique.umons.ac.be/genlog/SE/SE-contents.html`, and Wiktionary.

[6]Note that 6 of the 7 occurrences of "reality" in glossaries relate to a single definition. All other words occurred at most twice in a single definition.

[7]The demographics tables in the paper are missing, but the data underlying the tables is available on-line, `http://www.eecs.uottawa.ca/~tcl/edrel/EdrelData1998.xls`.

[8]Here we exclude general topics, such as data structures and design of algorithms; although they may well include software modeling, this is not the primary focus of the topic. Furthermore, modeling coverage is likely to be fairly "low level", and therefore not of as great relevance to the current study.

(although with different ranking). This can be summarized with the following list that combines the topics found to be important within the three studies.

- analysis and design methods
- requirements development
- requirements gathering/analysis
- software architecture
- software design and patterns
- software design patterns (possibly same as above?)
- software design practices

It is worthwhile to note which of these topics were considered to have a wide "knowledge gap" – those topics which were considered important but which were not covered sufficiently in undergraduate coursework. In the Lethbridge study these topics were software design and patterns, software architecture, requirements gathering/analysis, and analysis and design methods. In the Kitchenham et al. study these were: requirements gathering/analysis as well as software design patterns. Finally, von Wangenheim and da Silva found these to be: requirements development, software architecture, and software design and patterns. This variation is quite interesting. However, it should be noted that these three studies were conducted at different times, and with different selection criteria for the participants. It is thus not surprising that the topics considered under-taught varied.

In an interesting and relevant to this study series of papers [32, 33, 34] Zendler et al. consider not only concepts in computer science education, but also skills (process) and the combination of concepts and processes, i.e. the application of knowledge. In one of their papers [34, 393] they write,

> the content concept *model* is ranked at a modest 13 – below the content concepts *communication*, *language* and *test*. By involving process concepts, *model* has been upgraded significantly. *Model* is now ranked at 3 and closely follows the content concepts *problem* and *information*, which occupy the first two positions. Thus, activities around *dealing with models* are much more important than having pure knowledge about the content concept *model*.

From this it seems clear that while modeling is considered an important topic in the education of future computer scientists, it is not represented explicitly or extensively enough in computing curricula.

4.4.2 Topics identified by survey respondents

Since the previous "topic" studies did not concentrate on software modeling, the current study involved such explicit questions asked to the respondents of the survey. The respondents did not have a closed list of topics to choose from. Instead, they could freely define their topics. This resulted in several recurring themes but also in some interesting individual remarks. It has to be remembered that the responses were made by very experienced practitioners and the reported topics are based on their practice in teaching modeling. Since there was no predefined list of topics, it is difficult to come up with such list based on the survey responses. Instead, some linguistic analysis is presented below with a summary discussion on the responses.

Most Important Topics.

The following phrases related to software modeling, worth emphasizing were used by the respondents: essence, application, accuracy, purpose, understanding, real life, domain, balance, rigor, abstract thinking, abstraction, communication, different views, model building, modeling as a means, case studies, real software development, formal processing, automation, transformation.

In summary, while most respondents considered abstraction important, some focused on welding abstraction to implementation and the practical nature of modeling. Modeling is employed to help students stand back from the minutiae of systems and programming structures, statements and system linkages so that they can clearly see and respond to a task at hand both as a domain and a task that belongs to a domain. Therefore, modeling is seen to serve the kind of problem solving that defines computer science and software engineering and the important topics are those that accomplish that goal.

Least Important Topics.

The following phrases were found worth emphasizing in the respondents' responses to the question of least important topics: symbols, tools, notation, syntax, methodologies, technical details, metamodels.

In order to focus on the use of abstractions, respondents found that time and effort should not be wasted on such issues as language syntax and related tools. Some even pointed that UML language details often failed to clarify models, even though they were developed to accomplish just the opposite. Other areas respondents said that they did not consider useful for students learning how to model include pictographic languages, risk analysis, notation, and formal underpinnings of systems. It can be observed that the list of unimportant topics is quite short as the same topics recur in several responses.

Topics Lacking in Current Courses / Curricula.

The following phrases can be emphasized as reported lacking by the respondents: hard to say, none, DSL, MDD/MDA, practical, benefits of modeling, convincing samples, theoretical foundations, good textbook, integration (how it fits together), teaching methods.

Several respondents did not find any topics lacking in the curricula they represent. Other respondents indicated that without a Domain-Specific Language (DSL) for modeling, teaching it was tricky. More tools for teaching modeling is cited by many respondents as desirable. Some faculty re-

Table 3: Important modeling topics within 18 syllabi.

Tier	Topics
$\geq$ 10 occurrences	UML diagrams
4–9 occurrences	design patterns relationships object oriented model class
2–3 occurrences	architecture state machine interaction load (performance modeling)

quested more convincing examples, special purpose notation and grounding in Model-Driven Development and Applications. Formal development methods and the modeling of business development were also cited as lacking.

4.4.3 *Topics from course syllabi*

The survey responses on the important modeling topics included more generalized opinions of the respondents. This had to be thus complemented by a more detailed list of topics that were in fact taught in the curricula. The study was based on the 18 collected syllabi. It has to be stressed that the syllabi was collected from different universities that those represented by the survey respondents. It is interesting to note that the topics listed in syllabi vary somewhat, but the topics covered by most courses included UML, classes, and diagrams. This suggests that the learning toolset involves the use of modeling notations including class diagrams and how different components in a model are related and interact. Several courses specified object-oriented topics in general and specific manifestations such as design patterns and state machines. Likewise relationships, interactions, and architecture were specified as topics.

The topics can be placed on three tiers based on occurring more than once. Table 3 shows the results for 3 tiers, based on frequency of topic occurrence in our 18 syllabi. As it was just mentioned, the teaching of UML and diagrams is very common. Many of the topics are directly related to software design (patterns, architecture etc.). Other topics in the courses were of a more general computer science nature or specific to a topic that an institution or professor included. An important observation is that the topics do not explicitly include such topics like abstract thinking, communication, domain modeling that were reported as important by the survey respondents. What is more, it seems that the emphasis is put on the notations (UML) although that was reported as the least important aspect.

5. DISCUSSION

The results in Section 4 show a picture of the current status of how software modeling is taught. This gives a basis to discuss the four research questions defined in Section 3.

Referring to question 1, we may conclude that the current computing curriculum recommendations do not to put much emphasis on software modeling. It is rarely treated as a core knowledge area. Most often, the recommendations use the terms "model" or "modeling" without defining them. This can cause considerable confusion when designing a computing course. We feel this is a serious flaw in the overall approach to teaching software development, including programming. As computer programming deals with increasingly complex problems that students (and later – professionals) must solve, the need for modeling increases. Object-oriented modeling as well as other modeling techniques have sought to reduce complexity by abstracting larger problems into understandable and solvable units. In doing so, the requirements and style of programming changes from essential algorithms to solutions taken in a larger context defined by the models. While learning algorithms continues to be important, they are not enough by themselves without a larger model that gives them the ability to deal with the complexity that defines computer programming.

The current state-of-practice (research question 2) seems to include at most one or two (often elective) courses associated with modeling somewhere at the higher years of university education. Modeling is definitely treated as an advanced topic. This might be caused by the complexity and broadness of the "unified" approaches (UML), and due to modeling often being connoted with model-driven development and complex automated transformation approaches and languages. It seems to be very rare that modeling is treated as a knowledge area that recurs in many (including those foundational) courses in a coherent way. Even the curricula described and influenced by experienced modeling teachers seem to lack this coherence. Again, we argue that this is a serious flaw. Currently, students in their first years of computing education are taught how to program and work with algorithms that together form programs of limited complexity. When (and if) models and modeling are introduced later in their education, they have to re-learn programming because they cannot efficiently solve more complex problems with the programming practices they have learned up to the point where modeling is introduced. Under the current educational practices in software engineering, students first learn to program in one way – without a model – and then they learn how to program using models. This practice is expensive, inefficient and fundamentally wrong. The current practice is one of learn, unlearn, and re-learn programming.

Another fundamental problem is associated with research question 3. The analysis clearly shows high degree of confusion and incoherence in defining the scope of software modeling to be taught. In our opinion this calls for immediate improvement. A common understanding of the knowledge area should be reached through formulating clear definitions of software modeling. What is important, the definitions should reflect the importance of teaching abstract thinking *before* or *parallel* to teaching algorithmic thinking (programming). In line with this definition, modeling should be taught as soon as computer programming is taught. In this way, the students learn how to program in a way that will help them solve larger and more complex problems that they are likely to encounter upon graduation or entry into graduate school. They save time in that they do not have to unlearn an initial process, the process is less confusing since model learning does not conflict with non-model learning; they have never dealt with non-model programming. Learning to program

without knowledge of abstraction mechanisms, associated with modeling techniques can be argued as a serious flaw in contemporary approaches to teaching computing in general. This can be compared to gaining bad habits associated with learning to program using the "goto" statement.

It has to be stressed that model-enhanced programming never diminishes the importance of algorithms. They are simply taught and used in a modeled context and environment. The solutions of the small problems are integrated into systems that can integrate the small solutions into a whole that effectively handles the complex problems of programming. What is also important, the students can also easily associate solution models with the problem domain models. This way, modeling is treated as a means to understand a problem, just like it is in other engineering disciplines, and even generally – in everyday life (cf. eg. a globe model to determine Earth distances). This is strictly associated with the research question 4. It can be noted that the experienced modeling educators agree that modeling is about abstract thinking and understanding/communication. This seems to be in contrast to how modeling is taught in reality. Most often, it is associated with teaching the students the syntax and some semantics of a modeling language (usually UML). This is sometimes done in the context of software design or general software engineering methodologies. The students usually cannot appreciate the role of modeling, as it is not used in practical situations. Instead, and in line with what experienced modelers say, we argue that the courses of abstract thinking through constructing models should start as early as possible in the curriculum. It is also crucial that students gain understanding about how models of a problem domain (understandable to "ordinary laymen", including school children) can be translated and reformulated into the models of a problem solution, leading to code.

6. SUMMARY: RECOMMENDATIONS

The data collected through this research raises several important questions associated with the teaching of software modeling. This discussion leads to promoting the following specific recommendations:

1. **Define software modeling.** Software modeling is a significant knowledge area in the field of computing and must be given a clear definition. A clear and commonly accepted definition is important for constructing concrete curricula and curriculum recommendations. The current study provides data that allows to formulate a common ground definition. This report provides a proposal for such a definition in Appendix D. Moreover, there is no reason that the definition of software modeling cannot be revisited and revised throughout the curriculum, and given greater breadth and depth as the students' own understanding of the topic grows.
2. **Teach modeling early.** Software modeling must be taught early on in the curriculum. Generally, software modeling is taught as an advanced topic. This way, the students first learn the details of different software development activities, and only after learn how to deal with these activities using models. However, the German secondary school recommendations (GI) demonstrate that teaching software modeling across levels is viable. Students are exposed to and use models in many subjects prior to the post-secondary level, such as in literary or dramatic structure, maps in geography, equations in physics, and even plays in sports. There seems no reason that software models cannot leverage this knowledge of models sooner and more explicit than in advanced undergraduate courses. Although it may not be feasible for students to initially develop their own software models, they can learn to use, interpret and apply existing models (e.g., as described in [3, 15]).
3. **Teach modeling throughout.** Software modeling must be taught throughout the curriculum. Software modeling is a broad and deep area, and while it can and should be introduced early, there are many advanced topics in modeling (e.g., metamodeling, model transformation, model-driven architecture (MDA), domain-specific languages (DSL)) for which students need some maturity.
4. **Teach modeling together with programming.** Software modeling is relevant to other topics students study, such as programming. It is not pedagogically sound to teaching programming in the absence of modeling, only to have to re-learn programming after having learned about modeling (e.g., as described in [3]).
5. **Teach modeling foundations (notation *and* semantics).** Students' knowledge of models must be built on a solid and formal foundation in order for them to grasp advanced topics, such as model-to-model transformations. A first step might be to learn the syntax and semantics of a simple modeling language in order to be able to read and interpret such models. Later on, more elaborate modeling languages can be introduced, and students can learn to evaluate different models with respect to various criteria.
6. **Convey the practical applications of modeling.** The practical applications of software modeling must be clearly conveyed. Students who do not have a clear picture of the purpose and benefits of modeling might see modeling as "overhead" lacking practical value. Experiencing firsthand the benefits of software modeling is important. Such experience can, for example, be achieved by projects or internships (e.g., as in the MDE Diploma [9]).

These recommendations lead to the final recommendation to revise existing CS, SE and IS curricula and curricula recommendations regarding the inclusion of software modeling. Modeling should be considered a separate knowledge area dealing with ways to handle complexity throughout all phases of software and systems development. The curricula should treat modeling uniformly and coherently throughout the whole course of studies. This is very important, as modeling is an activity that is performed in all disciplines of software and systems engineering. Failure to present a coherent picture of modeling[9] to the students can lead to significant communication problems in various future real life situations.

[9]It is worth underlining that a coherent picture does not mean teaching a unified or single language, but it means teaching a coherent approach to dealing with complexity.

Acknowledgment

One of the authors of this research was partially funded by the EU within the REMICS project (contract number ICT-257793 under the 7th Framework Programme), see `http://www.remics.eu/`. Furthermore, this work is part of the BESQ+ research project funded by the Knowledge Foundation (grant: 20100311) in Sweden.

7. REFERENCES

[1] A. Abran, J. Moore, P. Bourque, and R. Dupuis. *Guide to the software engineering body of knowledge*. IEEE CS Press, 2004.

[2] H. Aris. Improved understanding through complete object-oriented software development example: An experience. In *Proceedings of the 4th Educators Symposium at MoDELS*, page 15p, 2008.

[3] J. Bennedsen and M. Caspersen. Programming in context: a model-first approach to cs1. In *ACM SIGCSE Bulletin*, volume 36, pages 477–481, 2004.

[4] J. Bezivin, R. France, M. Gogolla, O. Haugen, G. Taentzer, and D. Varro. Teaching modeling: Why, when, what? In *MODELS 2009 Workshops*, pages 55–62. Springer, 2010.

[5] M. Blay-Fornarino. Project-based teaching for model-driven engineering. In *Proceedings of the 4th Educators Symposium at MoDELS 2008*, page 15p, 2008.

[6] T. Bondesson. Software engineering education improvement. Master's thesis, Karlskrona, Sweden, 2004.

[7] J. Börstler. Improving crc-card role-play with role-play diagrams. In *Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications (OOPSLA)*, pages 356–364, 2005.

[8] J. Börstler and C. Schulte. Teaching object oriented modelling with crc cards and roleplaying games. 2005.

[9] J. Cabot and M. Tisi. The mde diploma: first international postgraduate specialization in model-driven engineering. *Computer Science Education*, 21(4):389–402, 2011.

[10] A. Cowling. Modelling: A neglected feature in the software engineering curriculum. In *Proceedings of the 16^{th} Conference on Software Engineering Education and Training*, 2003.

[11] A. Cowling. The role of modelling in the software engineering curriculum. *Journal of Systems and Software*, 75(1):41–53, 2005.

[12] I. Diethelm, L. Geiger, and A. Zündorf. Teaching modeling with objects first. In *Proceedings of the 8th World Conference on Computers in Education*, 2005.

[13] R. France. Teaching student programmers how to model: Opportunities & challenges. In *Keynote at the 7th Educators Symposium at MoDELS*, page 15p, 2011.

[14] J. Gilbert. Models and modelling: Routes to more authentic science education. *International Journal of Science and Mathematics Education*, 2(2):115–130, 2004.

[15] I. Hadar and E. Hadar. An iterative methodology for teaching object oriented concepts. *Informatics in Education*, 6(1):67–80, 2007.

[16] Joint ACM/IEEE Task Force on Computing Curricula. Software engineering 2004: Curriculum guidelines for undergraduate degree programs in software engineering. Technical report, IEEE CS and ACM, 2004.

[17] M. Khine and I. Saleh, editors. *Models and Modeling: Cognitive Tools for Scientific Enquiry*, volume 6 of *Models and Modeling in Science Education*. Springer Verlag, 1 edition, 2011.

[18] B. Kitchenham, D. Budgen, P. Brereton, and P. Woodall. An investigation of software engineering curricula. *Journal of Systems and Software*, 74(3):325–335, 2005.

[19] J. Kramer. Is abstraction the key to computing? *Communications of the ACM*, 50(4):36–42, 2007.

[20] T. Kuehne. What is a model? In *Language Engineering for Model-Driven Software Development, Dagstuhl Seminar Proceedings 04101*, 2005.

[21] L. Kuzniarz and J. Börstler. Teaching modeling—an initial classification of related issues. In *Pre-Proceedings of the 7th Educators' Symposium@MODELS 2011 – Software Modeling in Education*, pages 65–70, 2011.

[22] L. Kuzniarz and M. Staron. Best practices for teaching UML-based software development. In *Satellite Events at the MoDELS 2005 Conference*, pages 320–332, 2006.

[23] T. Lethbridge. What knowledge is important to a software professional? *Computer*, 33(5):44–50, 2000.

[24] T. Lethbridge, R. LeBlanc, A. Sobel, T. Hilburn, and J. Diaz-Herrera. Se2004: Recommendations for undergraduate software engineering curricula. *IEEE Software*, 23(6):19–25, 2006.

[25] J. Ludewig. Models in software engineering – an introduction. *Software*, 2:5–14, 2003.

[26] P. Muller, F. Fondement, B. Baudry, and B. Combemale. Modeling modeling modeling. *Software and Systems Modeling*, pages 1–13, 2010.

[27] A. Pyster. Software engineering 2009 (GSwE2009): Curriculum guidelines for graduate degree programs in software engineering. Technical report, Stevens Institute of Technology, 2009.

[28] P. Roberts. Abstract thinking: a predictor of modelling ability? In *Proceedings of the 5th Educators Symposium at MoDELS*, page 12p, 2009.

[29] C. Schulte and J. Niere. Thinking in object structures: Teaching modelling in secondary schools. In *ECOOP Workshop on Pedagogies and Tools for the Learning of Object-Oriented Concepts*, 2002.

[30] J. Vallino. If you're not modeling, you're just programming: Modeling throughout an undergraduate software engineering program. In *Proceedings of the 2nd Educators Symposium at MoDELS*, page 15p, 2006.

[31] C. von Wangenheim and D. da Silva. Survey on the relevance of topics in computer science education. Technical report, University do Vale do Itajai, San Jose, Brasil, 2009.

[32] A. Zendler and C. Spannagel. Empirical foundation of central concepts for computer science education. *Journal on Educational Resources in Computing*, 8(2):6:1–6:15, 2008.

[33] A. Zendler, C. Spannagel, and D. Klaudt. Process as content in computer science education: empirical

determination of central processes. *Computer Science Education*, 18(4):231–245, 2008.

[34] A. Zendler, C. Spannagel, and D. Klaudt. Marrying content and process in computer science education. *IEEE Transactions on Education*, 54(3):387–397, 2011.

APPENDIX

A. SURVEY: CALL FOR PARTICIPATION

I'm participating in an ITiCSE working group on the teaching of software modeling. Since you are experienced in this area, I would like to ask you a few questions, which will be used as input to our investigation of the current state-of-the-art in the area of teaching software modeling in SE/CS/IT curricula.

The working group's mission is described as follows, see the working group web page at `http://www.iticse12.org.il`:

> The development and proper usage of conceptual models, diagrams, or workflows are important learning objectives in any computer science related curriculum. However, there is very little systematic work on the role of modeling in SE/CS/IT education beyond OOA/OOD/OOP or particular modeling languages and tools. Our primary interest is in conceptual modeling, domain modeling, software design, etc. Mathematical modeling or simulation or modeling in science/engineering in general is out of the scope of the working group.

If you could take a few minutes to answer the following questions (10 in total), we would be very grateful. We believe you are well-qualified to provide us with input about the teaching of modeling. The purpose of this interview is to get *your* perceptions and *your* experiences; there are no right or wrong or desirable or undesirable answers.

All participants will be anonymized and all information you provide will be kept confidential.

Note: With "modeling course" we refer to any course that contains some unit(s) on modeling or modeling related subjects. It doesn't need to be a course specifically dedicated to modeling.

Q1: Please provide some information about your background that might be relevant (years of experience, type of experience (education/research/industry)).

Q2: In a few words, what would you say is modeling in the context of software development?

Q3: In which courses that you teach/taught/developed is/was modeling instruction part of the course? What were the modeling units/topics?

Q4: Which other modeling course are taught at your school?

Q5: How are these different from yours and/or each other?

Q6: What do you think is most important to teach in modeling?

Q7: What is least important?

Q8: Is there anything lacking in those courses – any modeling topic that is not taught but should be taught?

Q9: Do you think there is enough support for teaching modeling (Methods, languages, tools, etc.)?

Q10: Anything more you want to add/say?

Thank you very much in advance for your help.

B. LIST OF CURRICULUM RECOMMENDATIONS

CS2013 Curriculum Guidelines for Undergraduate Degree Programs in Computer Science (ACM/IEEE) (Strawman version) – available from `http://cs2013.org/`

GI Grundsätze und Standards für die Informatik in der Schule, Bildungsstandards Informatik für die Sekundarstufe I (German Computer Society) – available (in German) from `http://www.gi.de/service/publikationen/empfehlungen.html`

GSwE2009 Graduate SE Curriculum Guidelines for Graduate Degree Programs in Software Engineering (ACM/IEEE) 2009 – available from `http://www.acm.org/education`

IFIP Informatics Curriculum Framework 2000 for Higher Education (IFIP, UNESCO) – available from `http://www.ifip-tc3.net/`

LACS A 2007 Model Curriculum for a Liberal Arts Degree in Computer Science Liberal Arts Computer Science Consortium – available from `http://www.lacs.edu/`

MSIS2006 Model Curriculum and Guidelines for Graduate Degree Programs in Information Systems (ACM/IEEE) – available from `http://www.acm.org/education`

PLCS Teaching Minima for Informatics Courses (Polish Ministry of Science and Higher Education) – available (in Polish) from

SE2004 Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (ACM/IEEE) – available from `http://www.acm.org/education`

C. SECTIONS FOR REVIEWING CURRICULUM RECOMMENDATIONS

The following structure was used for curriculum reviews.

- Overview
- How broad or narrow is modeling understood/ described/ defined in the document?
- Are there any implicit or explicit statements about inclusion or exclusion of certain topics, i.e., things that are considered to be modeling or not? If so which are these topics?
- As how important is modeling perceived in relation to other topics?
- Personal reflections

D. A DEFINITION OF SOFTWARE MODELING

The definition below is an attempt in integrating the definitions analyzed in this study. A commonly accepted definition of modeling would be an important stepping stone for further discussions on the role of software modeling in computing curricula. The definition is divided into four sections that cover the four dimensions of modeling described in Section 4.3.

Why. The purpose of modeling is to support the software development process with methods, languages, and tools for making complex problems or software artifacts easier to understand, communicate, and process.

What. Software models are graphical or textual descriptions of some reality associated with software, including the domain that software supports. Depending on their purpose, the models can vary in their degree of formality and suitability for automatic processing. Software models capture knowledge in two areas: 1) domain knowledge and 2) software construction knowledge. Software models are expressed with modeling languages that have their syntax and semantics defined to the level of formality necessary to capture and process that knowledge.

How. Software modeling deals with taming complexity by abstraction and decomposition. Software models concentrate on those elements or issues that are relevant to a particular purpose, question, or view. These models can be on various levels of abstraction. Information presented in different models should be consistent. Depending on the purpose, software modeling utilizes tools starting from pen-and-paper approaches and ending at formal model checkers and model compilers.

Where. Software modeling is a set of activities to facilitate understanding of various issues associated with all phases of software development, through creating software models.