The 30th IEEE Conference on Software Engineering Education and Training

Model Driven Software Engineering in Education: A Multi-Case Study on Perception of Tools and UML

Grischa Liebel
Chalmers | University of Gothenburg
Gothenburg, Sweden
Email: grischa@chalmers.se

Omar Badreddin
Computer Science Department
University of Texas, El Paso, USA
Email: obbadreddin@utep.edu

Rogardt Heldal
Western Norway University of Applied Sciences
Bergen, Norway
Email: rogardt.heldal@hib.no

***Abstract*—While several benefits of using models in software engineering have been observed in practice, the adoption of modeling remains low. Multiple challenges of using models, especially related to tools, have been reported both for industrial use and for education. However, there is a lack of systematic, empirical investigations of the challenges in modeling education and their relation to industrial challenges. Therefore, we conducted a multiple-case study with two cases, in the U.S and Sweden, focusing on students' perceptions towards tooling and UML in education. Our data collected from 369 student evaluation surveys, enriched with qualitative data, shows that the students' perception of modeling tools depends not only on the complexity of tools, but rather on multiple contextual factors, including tool characteristics, scope of course and project contents, nature of the required models, and the tools' role in generating executable artifacts. We conclude that there is a need for tailoring modeling tools for education beyond focusing on simplification and usability. Furthermore, due to the broad diversity within the modeling domain, there is a need for adapting the use of tools to the specific curriculum and course learning objectives.**

***Keywords*-UML, Model-Driven Architecture, Model-Driven Engineering, Pedagogy.**

I. Introduction

Software modeling is recommended in several computer science and software engineering curricula [1]. Furthermore, it has been shown that creating up front design improves the overall quality of the software under development, and can facilitate automatic generation of various system artifacts [2].

However, while modeling and model-driven practices such as MDA [3] or MBE (see [4]) are widely adopted in some domains like embedded systems [5], other communities such as the open source and agile communities remain almost exclusively code-centric [6]. Other studies indicate that the use of models is low in software engineering and mainly limited to sketches for communication and coordination [7].

Several challenges to the adoption of modeling in practice are reported by empirical studies, e.g., that modeling is not perceived useful enough [8] or that it is only useful for very complex systems [7]. Additionally, it has been reported that the adoption of model based engineering of software is hampered, at least in part, by inadequate academic preparation of young software engineers [9]. This is supported by evidence that students graduate from software engineering and computer science programs perceiving UML and model driven methodologies as ineffective [10].

Interestingly, industrial use of models and modeling education seem to be facing one common challenge, namely tooling. Many of the reported challenges for industry adoption are related to the tools used for modeling, including inadequate usability [11], [5], interoperability [12], [5], [13], and complexity [12], [14]. Similarly, there exists anecdotal and empirical evidence that tooling plays a decisive role in modeling education [15], [16], [17], [18].

Due to the challenges arising from using industrial modeling tools in the classroom, several educators have argued for simplified tools in education, e.g., [19], [20], [21], [15]. Others have argued that industrial-grade modeling tools can be used given the right support and should in fact be used in order to address the lack of adoption in industry [17]. However, while there is substantial anecdotal evidence, empirical evidence on the influence of modeling tools and tool support on the students' perception of modeling is lacking. Therefore, existing conclusions on the need of educational tools are likely to be influenced, at least to some extent, by personal preconceptions, i.e., they may suffer from confirmation or self-serving bias.

To address the lack of empirical evidence, we present in this paper a multiple-case study with two cases focusing on tools and their support in modeling education. We aim to answer the following research questions.

RQ1: How do students perceive modeling tools?
RQ2: How does the course context and the use of modeling tools influence the students' perception of modeling?
RQ3: Compared to industrial use, what characteristics do modeling tools need when used in education?

We collected quantitative and qualitative data from undergraduate students in two courses over two years in the U.S. and in Sweden. To minimize confirmation bias and to ensure a balanced interpretation of the collected data, the authors of this paper come from two different camps; proponents of simplified modeling tools in education and opponents thereof.

The paper is organized as follows. We present a background and related works in the next two sections. We present our multiple-case study design in Section III. The data collected from the multiple-case study is presented in Section IV. We present a discussion of the results in Section V. In Section

VI, we discuss threats to validity. We conclude the paper in Section VII.

II. Related Work

A. State of Modeling in Software Engineering Education

Computer Science and Software Engineering curricula focus on programming and coding as the main problem-solving tool, even though several curriculum recommendations promotes the use of modeling [1]. Today, the majority of the curricula introduce modeling and design concepts only lightly in early courses. Model Driven Architecture (MDA) concepts are typically introduced as advanced selective topics later in the curriculum. In a previous work, Badreddin et al. conducted a longitudinal study of seven Computer Science and Software Engineering programs at four Universities [10]. The study focused on students' perception of modeling and MDA as they progressed in their education from early bachelor years and to their terminal educational degrees. Among other findings, the study identified a consistent downward trend in perception of MDA effectiveness in software development. Students graduate with increasing conviction that MDA is not an effective software development methodology. This is potentially due to the fact that overhead of model creation is not justified in the case of small course-sized problems.

B. Educators' Experiences

Experiences with the use of modeling tools of different complexity and maturity, for different purposes, are a common topic in education research.

Industry-grade tools are often described as unsuited for education in the area of modeling. For example, Lethbridge et al. [15] describe industry-grade tools as "unwieldy". Similarly, Akayama et al. [16] report from their experience that students use language features they do not understand when using industry-grade modeling tools. Paige et al. [18] describe industry-grade tools as too cumbersome. While all of the above publications are based on extensive teaching experience, they lack empirical data that substantiate their statements.

Cabot and Kolovos [22] present their experiences[1] along with student evaluation data on two failed attempts to introduce MDE to students. The authors emphasize the importance of a first positive exposure to MDE, but also refer to several shortcomings in tooling, such as a lack of documentation.

Liebel et al. [17] report based on evaluation data from two years of a modeling course at the Bachelor level that industry-grade tools can be used successfully in the classroom, given sufficient and dedicated tool support.

How the use of tools compares to not using any tools at all in modeling education is studied by Hammouda et al. [23], who compare the use of modeling tools and pen and paper modeling in education. The authors do not observe a clear advantage for any of the two approaches.

Hence, while it is often claimed that industry-grade tools are unsuited for education, there is lack of empirical evidence beyond experience reports that substantiates or refutes this claim.

[1]While the authors call it an "experiment", they do not perform a controlled experiment in the classical sense.

TABLE I
Overview of education-focused tools

Tool	Key Advantage
QuickUML [25]	Limited features to ensure students use only features they understand.
minimUML [26]	Support for undo/redo to encourage explorative learning.
Violet [27]	Lightweight for early learners.
UMLet [21]	Simple interface to facilitate fast UML models creation.
Dia [28]	Simple drawing tool, easy to learn.
ArgoUML [20]	Full featured to support system development.
StudentUML [19]	Limited support for a small subset of UML.
Ideogramic UML [29]	Support for interactive learning by transforming gestures into formal UML models.
Umple [15]	Textual modeling for seamless integration into OO code.
MDELite [30]	Teaching UML from a relational data base perspective.
UMLFactory	UMLFactory referenced at www.UMLfactory.com is no longer available.
yUML [31]	Cloud based for sharing and collaboration.
TxtUML [32]	Textual executable modeling.
WebUML [33]	Web based for sharing of models.

C. Overview of Education-Focused Tools

The question of tool choice in education has been raised just after the emergence of UML and its standardization. One of the early concerns that have been raised at that time was the adequacy of existing computing platforms for running commercial modeling tools that had high memory requirements [24]. Typical concerns are the complexity of modeling tools, the steep learning curve, and the complexity of the UML standard itself.

These concerns seem to have been a common motivation for the majority of specialized UML tools in existence today, despite the fact that industry is seemingly facing similar challenges, and despite the lack of empirical evidence that show any disadvantages of industrial modeling tools.

In Table I, we give an overview over a number of education-focused UML modeling tools and their key advantages. We chose these tools based on their exposure in the academic modeling and MDE community, e.g., through existing publications.

III. Method

Our method is a multiple-case study with four units of analysis. Case studies are appropriate where the object of study is difficult to separate from its real-life context [34]. In the case of university education, there are many factors that are very difficult to control, if not impossible. For example, lecture quality, students' knowledge and preparation, social influence, variations within students' populations over different course offerings, and variations in project assignments and assessments.

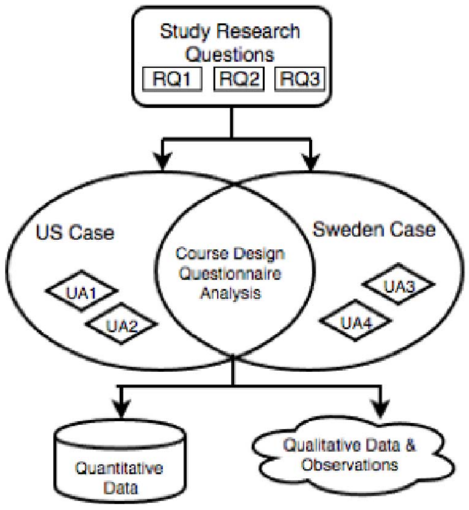

Fig. 1. Case study design

We conducted a multiple-case study with two cases and four units of analysis according to the classification in [35]. The cases are two third-year courses on software modeling and design, one conducted in the USA and one in Sweden. The four units of analysis (UoA) are the different years in which the course was run. In the US case, the course was run once with Umple [15] as a modeling tool and once with Papyrus [36]. In the Sweden case, two course runs with Papyrus were studied, once with dedicated tool support and once without. Both cases contained a group project with a final presentation. However, group sizes varied from three to four students in the US case, and up to eight students in the Sweden case. The course lasted 15 weeks in the US case and 8 weeks in the Sweden case. Otherwise, conditions were similar over the different units of analysis. For both cases, quantitative data and qualitative data were collected. Quantitative data are collected by means of anonymous and voluntary questionnaires conducted towards the end of the course. Qualitative data is collected by means of one-on-one follow-up questions, and from the end of term course evaluations.

A. Case Context

In both cases, we applied constructive alignment [37]. That is, our students apply more than 80% of the material covered during lectures in their projects[2]. Also, the students create both behavioral and structural models. The final deliverables must include a running system. This course design ensures that the students receive some level of automated feedback on their models, e.g., using code generation or executable models, that helps them to understand implications of designs they have created. When choosing a project topic, we did consider realistic topics from industry, as suggested by [18]. However, we ultimately decided against an industrial project topic due to time constraints and to lower complexity. We instead prescribed a topic with minimum requirements. Students are then required to extend the topic by conducting requirements elicitation, analysis and design.

[2]Other content being guest lectures or motivation material not directly related to the project.

B. Data Collection

We collected quantitative[3] and qualitative data through a paper-based questionnaire at the end of the course, in order to ensure that students' responses are based on their experiences with modeling and the assigned tool. Participation was both voluntary and anonymous. In total, we collected 38 evaluation surveys in UoA 1, 39 in UoA 2, 151 in UoA 3 and 141 in UoA 4. The following two subsections describe the quantitative and qualitative data collected.

1) Quantitative Data: Anonymous questionnaires were distributed to students towards the end of the course. In all four UoAs, students were asked to rate their agreement to the following core questions.

Q1. <Tool> can be useful for modeling a large and complex system.
Q2. The use of <Tool> affected our project positively.
Q3. It was easy to learn <Tool>.
Q4. UML can be useful for modeling a large and complex system.
Q5. I observed benefits of creating models of our system prior to writing the code.
Q6. The advantages of modeling outweigh the creation effort of the models.

Each question had 5-scale Likert choices: Strongly Agree, Agree, Neutral, Disagree, Strongly Disagree, and Don't know. Missing and Don't know answers are excluded from the graphs in the following.

2) Qualitative Data: Qualitative data was collected from open-ended questions and additional comments space provided in the questionnaires and, in the US case, from one-on-one discussions and interviews with students related to UML or the tool. Students voluntarily opted for the post-questionnaire interviews, by either writing their email addresses, or by showing up at pre-set scheduled time during the last two weeks of the semester. Additional qualitative data was collected from end of term course evaluations. We coded the qualitative data loosely following a grounded-theory approach [38].

IV. Results

In this section, we present the quantitative data results for both cases. The results are presented as follows. For each case, data for both its units of analysis are presented together. Data is presented under two classification; tool related data and UML related data. Qualitative data are used for interpreting the results in Section V.

[3]The quantitative raw data is published at http://grischaliebel.de/data/research/LBH-MBE-edu.zip

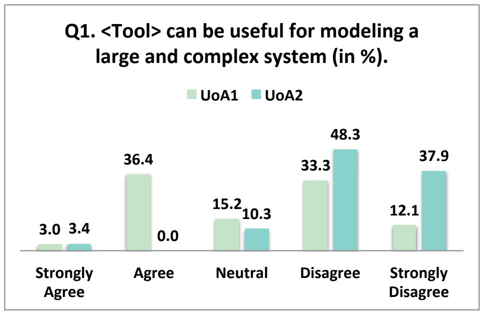

Fig. 2. Tool's usefulness for creating large systems (Case I)

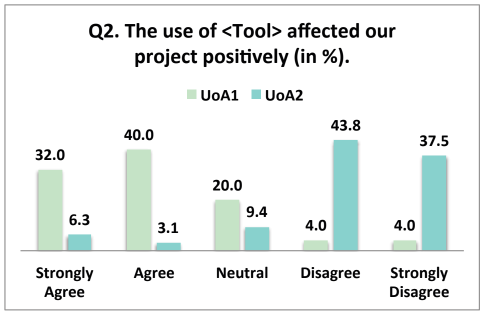

Fig. 3. Tool influence (Case I)

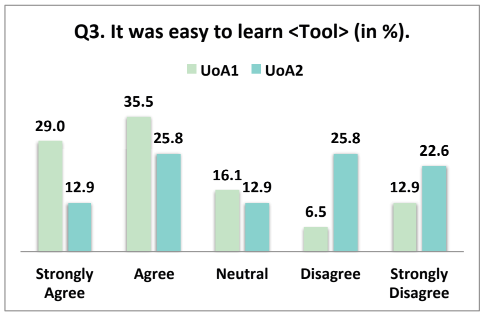

Fig. 4. Easiness to learn tool (Case I)

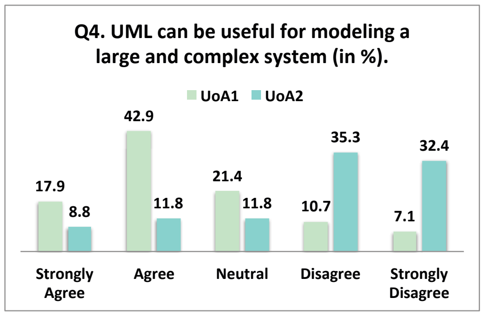

Fig. 5. UML's usefulness for creating large systems (Case I)

A. Case I: The US Case

In this case, students were instructed to use a modeling tool to implement their project (Umple [15] in UoA 1 and Papyrus [36] in UoA 2). They could choose from a list of pre-defined topics, e.g., a university registration system or a canal lock system. Students were instructed to create structural models (class diagrams) and behavioral models (state machines), and provide implementations of actions using Java. The project high-level requirements were presented in week two of the courses. Students presented their project design and implementation in week fourteen of the course, and submitted a final project report in week fifteen.

Students were presented with a demonstration of the tool by the course instructor, and were provided with optional two hands-on lab sessions given by the course teaching assistant.

1) Tool Data: We asked the students to rate their agreement to three statements regarding the use of the tool (Umple in UoA 1 and Papyrus in UoA 2), as discussed in Section III-B. The results are depicted in Fig. 2, Fig. 3, and Fig. 4.

In UoA 1, the students' views regarding the usefulness of the tool are somewhat balanced, with 40% of the students agreeing that Umple is useful for the development of large and complex systems and 46% disagreeing. In UoA 2, the view is significantly worse. Here, only 3.4% agree that Papyrus is effective for large and complex systems and 86% disagree.

There is a clear endorsement for Umple in UoA 1 regarding its impact on the project, with 65% agreeing that the tool affected their project positively and about 20% disagreeing. In UoA 2, the picture is similar to the previous question. Only 10% agree that Papyrus had a positive effect, while 81% disagree.

In UoA 1, 65% agree and 19% disagree that Umple is easy to learn. The view in UoA 2 is again more negative, with 39% agreeing and 48% disagreeing that Papyrus is easy to learn.

2) UML Data: We asked the students to rate their agreement to three statements regarding the UML, as discussion Section III-B. The results are depicted in Fig. 5, Fig. 6, and Fig. 7.

About 60% of students agree in UoA 1 that UML is useful for large and complex systems, while only 18% disagree. In UoA 2, this picture changes considerably to only 21% agreeing and 68% disagreeing.

Similarly, the view on benefits of creating models prior to writing code is almost mirrored in UoA 1 and UoA 2. While 78% agree that creating models before coding is effective and 22% disagree in UoA 1, only 30% agree and 59% disagree in UoA 2.

Compared to the previous two questions, the students' view does not change significantly between UoA 1 and UoA 2 regarding the modeling trade-off. In UoA 1, 21% agree that the advantages of modeling outweigh the creation effort, and about 64% disagree. In UoA 2, the picture is only slightly more negative with 13% agreeing and 70% disagreeing.

Overall, one observes that in UoA 1 the students clearly see the benefit of UML, but are somewhat sceptical regarding the effort involved. In UoA 2, this picture is changed completely towards a rather negative view of UML and modeling.

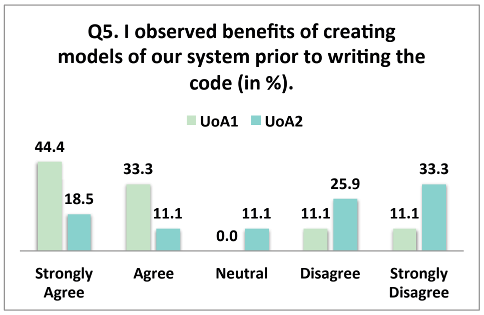

Fig. 6. Benefits of modeling (Case I)

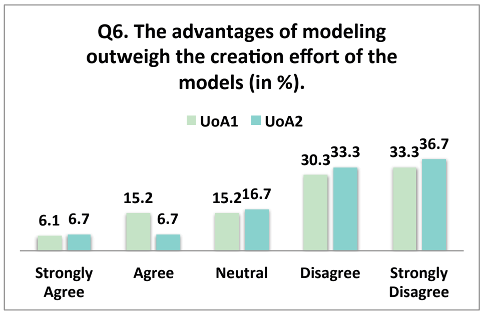

Fig. 7. Modeling tradeoff (Case I)

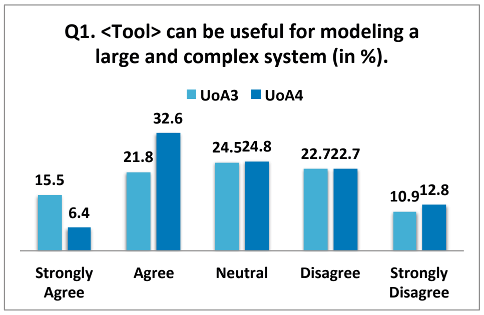

Fig. 8. Tool's usefulness for creating large systems (Case II)

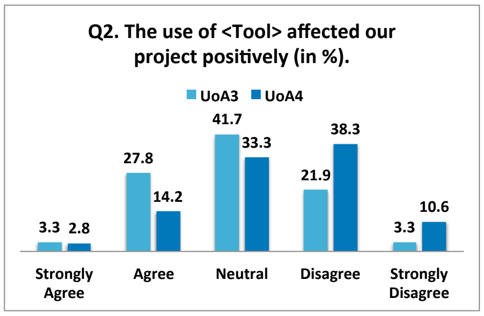

Fig. 9. Tool influence (Case II)

B. *Case II: The Sweden Case*

In this case, students used Papyrus to realize an 8-week project. During this project, students created in project groups several structural and behavioral UML models on different abstraction levels, e.g., domain models for the purpose of communication, class diagrams for design, or state machine diagrams to illustrate use cases. From the class diagrams, the students generated code using the Eclipse Modeling Framework [39] code generation capabilities and implemented a number of central use cases. All students are required to model and implement a hotel booking system, but only very vague requirements were stated initially, which the groups then had to refine and enrich with further features and/or requirements. Student groups had to attend weekly compulsory supervision sessions with an assigned supervisor.

Papyrus was introduced in two dedicated lectures, which were also recorded so that students could later re-watch the presentations. In UoA 3, one dedicated teacher knowledgeable in Papyrus and the course material provided dedicated support via email. In UoA 4, this additional support was not available due to a different allocation of teaching hours.

The results of the survey are presented in the following, separately for the three questions targeting Papyrus and those targeting UML.

1) Tool Data: We asked the students to rate their agreement to three statements regarding the use of Papyrus, namely the tool's effectiveness for development of large and complex systems (Fig. 8), the tool's usefulness in this specific project (Fig. 9), and how easy it was to learn the tool (Fig. 10).

The students' view of the usefulness of Papyrus to develop large and complex systems is different in the two UoAs. In UoA 3, about 15% strongly agree and 22% agree that Papyrus is useful for this purpose. In UoA 4, a similar proportion of students generally agree to the statement, but the percentage of students strongly agreeing is lower than in UoA 4 (only 6.4% strongly agree and 32.6% agree). Similarly, the percentage of students strongly disagreeing to the statement has increased by 2% to now 12.8%.

In UoA 3, there is no agreement among the students as to how useful Papyrus was for the project (31% (strongly) agree that it was useful, 25% (strongly) disagree). A much clearer picture can be seen for the statements in UoA 4, with 49% (strongly) disagreeing and only 17% (strongly) agreeing.

While not as strong as for the previous question, the picture for how easy it was to learn Papyrus is similar. In UoA 3, the students' evaluation is balanced with 29% (strongly) agreeing that it was easy and 33% (strongly) disagreeing. In UoA 4, this view has changed to the negative with only 11% (strongly) agreeing and 44% (strongly) disagreeing.

2) UML Data: The students' picture of UML is considerably more positive than their picture of Papyrus. We asked the students to rate their agreement to three statements regarding the UML, namely UML's usefulness for development of large and complex systems (Fig. 11), the benefit of creating models prior to coding (Fig. 12), and the tradeoff between modeling benefit and effort (Fig. 13).

In UoA 3, 86% of students agree that UML is useful for large and complex systems and only 1% of the students

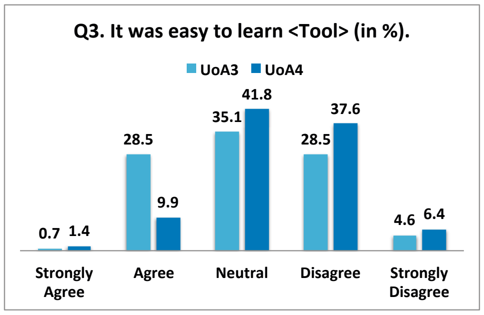

Fig. 10. Easiness to learn tool (Case II)

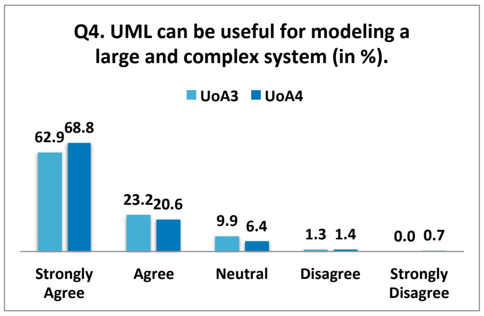

Fig. 11. UML's usefulness for creating large systems (Case II)

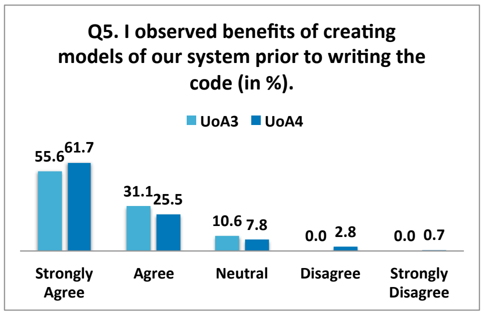

Fig. 12. Benefits of modeling (Case II)

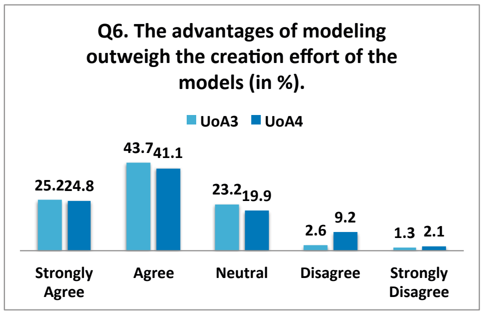

Fig. 13. Modeling tradeoff (Case II)

disagree. In UoA 4, the agreement increases to 89% of the students (strongly) agreeing that UML is useful for modeling a large and complex system.

Similar figures can be observed with respect to the benefits of creating models. In UoA 3, 87% agree that creating models before writing code was beneficial and not a single student disagrees. The same proportion of students agrees to the statement in UoA 4, but 3.5% disagree.

While still extremely positive, the views on the tradeoff between effort and benefit of modeling are slightly lower. In UoA 3, 69% agree that the advantages of modeling outweigh the creation effort, with only 4% disagreeing. Similarly, in UoA 4 66% (strongly) agree, but 11.3% disagree.

Overall, the evaluations of the UML statements can be seen as a rather strong endorsement of UML, especially given the balanced or negative evaluations of Papyrus. In particular, it is extremely interesting to observe that the evaluation of UML remains largely unchanged compared to UoA 3, despite the strong decline in the Papyrus evaluations.

V. Observations and Analysis

In this section, we provide observations and analysis of the entire data sets collected from both cases, including all four units of analysis. The observations and analysis are based on both quantitative and qualitative data sets. We then summarize this discussion in terms of our three research questions.

A. *Model Based Engineering versus Model Driven Engineering*

Both MBE and MDA involve creation and manipulation of models that abstract the system under development. However, in MDA, the focus is on system development and code generation. As such, models must have precise semantics to aid in code generation (as well as other artifacts such as test cases). In Case I where the focus was primarily on system development, two factors came into play; 1) tool choice and 2) overhead incurred to create and manage models. Our findings suggest that to support teaching of MDA effectively, the tool must support only a subset of UML with unambiguous semantics, and one that contributes directly to generating executable artifacts (code and tests). This could explain why Umple's overall perception was higher than Papyrus. In this context, students perceive UML as key in its support for code generation. Consequently, UML perception is greatly influenced by perception of effectiveness of the generated artifacts.

In Case II, the students' tasks involved, in addition to code generation, also modeling of requirements, analysis and design on a less formal level. As such, code generation becomes a secondary goal. This is where students tend to perceive UML more favourably. In fact, students' perception of the tool and UML are both positive when more informal modeling is involved. This finding is also consistent with a recent analysis of software engineering programs that demonstrates positive perception of UML when used for informal requirement gathering and analysis [10]. Hence, one possible conclusion resulting from this observation is that to improve students' view of modeling, a mix of informal and formal modeling should be included. Furthermore, as already discussed by Börstler et al. [1] in 2012, what is taught in *modeling* courses still differs greatly. As a result of this, researchers and educators should take great care in explaining their course context when publishing or discussing *software modeling*. As common terms in the area such as MDE or MBE can be ambiguous, a possible way to discuss course content could be the initial taxonomy of *what* is taught by Kuzniarz and Börstler [40].

B. Project Domain

In Case I, the project domain was familiar to students, to promote the focus on the system design and implementation. The problem domain and requirements were given, or assumed by the students. In Case II, only a vague problem description was given to the students, who then were required to define the domain and the requirements themselves. As such, there was initially a focus on exploration and understanding of the problem domain. This seems to have a positive impact on the students' perception of UML in general. As can be seen in the quantitative data, the UML perception of the students is very high in both UoA 3 and UoA 4. Additionally, the qualitative data supports this observation. For example, several students reported that UML is useful in defining requirements and achieving a common understanding of the problem within the project group.

This observation ties into the discussion on whether or not realistic problems should be used in student modeling projects. While Paige et al. [18] argue that realistic project topics are important to motivate students, we here (and earlier in [17]) take the standpoint that this imposes unnecessary complexity on students. However, the above observation suggests that the topic should encourage students to explore and gain further insights through the use of models. Hence, the topic needs to contain some uncertainty and complexity.

Finally, the suitability of the modeling tool(s) for the given project domain and scope might clearly have an influence on the students' perception. Clearly, a general-purpose UML modeling tool such as Papyrus is more suited in a project that includes the entire development chain including project scoping, requirements analysis, design, and code generation and implementation. In contrast, Umple might have been more suitable for a narrow project scope as in Case I, where the focus is on code generation and design models. Hence, it could be that the suitability of the tools in Case I for the given project has a larger impact on the students' perception than the tool quality, e.g., with respect to usability, itself.

C. Delivering a Working System and Support for Round-Trip Engineering

To develop a complete system, collaboratively, implies that models and code must be interchanged frequently. In UoA 2, the code generation facilities used by the students did not support effective re-generation and the students had to implement their own workarounds to avoid overwriting of their code in the face of changes in the design. Students would store code artifacts in separate files, and merge after iterative forward engineering (generating skeleton code from models). This seems to have significantly hampered the students' perception of UML and the tooling. In contrast, Umple, as used in UoA 1, tends to merge both model and code in a single artifact, significantly easing the need for forward and reverse engineering. This could explain the large differences in UML perception between UoA 1 and UoA 2.

In Case II, the students used the code generation facilities of EMF. While these are less than ideal for generating code from UML models, there is support for re-generation and protected regions in the code. Therefore, there is no need for workarounds to protect existing, hand-written code. This could explain why the students' view of Papyrus is much more positive compared to UoA 2. Additionally, we can tell from comments in the surveys that in Case II, several students appreciated the added value from re-generation after they had understood and mastered the process. This shows that while the tool does not need to be easy from the beginning, it has to provide the students with (perceived) added value by the end of their project. Otherwise, students may perceive it as cumbersome, and an obstacle in their way of being more effective.

D. Expert and Dedicated Tool Support

Dedicated tool support seems to have a substantial impact on students' overall perception. This can be observed in the comparison of UoA3 and UoA4. In UoA 3, there was dedicated support for Papyrus and code generation in Papyrus, including videos that students could watch repeatedly. The tool evaluation is consequently balanced, which could be seen favourably given the complexity of the tool and of code generation. More importantly, the change from UoA 3 to UoA 4 is statistically significant ($p < 0.01$ for all questions, Mann-Whitney U test [41]) in all three tool questions. Additionally, the dedicated support was emphasized in the qualitative evaluation data by a large amount of students in UoA 3. Given that the only change between the UoAs is the amount of tool support, this is a rather interesting result. It is essential to point out here that the tool support was in Case II given by a teacher within the course, not by the tool vendor or the community. As we dicsuss already in [17], we regard this as essential in order to give the students the *right* support for the given project. In fact, it could explain Cabot and Kolovos' [22] negative experience with tool support, as they seem to have relied mainly on the tool vendors in their study on teaching MDE.

Interestingly, the UML evaluation is consistent in the two UoAs and seemingly unaffected by the support. Given this result, it could be reasoned that the tool support is not detrimental, as the perception of UML -which is the part the learning outcome relates to- remains unchanged. However, if the support is insufficient, students might never reach the point of mastery where the tool provides added value, as discussed in the previous section. Additionally, we believe that a negative view of the tool might seriously hamper the students' learning, as discussed in the following section.

E. Students' Perception, Feedback, and Learning Outcomes

In two out of four UoAs, the tool evaluations can be viewed as being negative, in one UoA the tool evaluation is balanced and only in UoA 1 the tool evaluation is truly positive. This observation together with Hammouda et al.'s findings that there is no significant difference between using a tool and using just pen and paper for modeling [23] raises the question whether tools should at all be used in modeling education. As two of the co-authors of this paper already discussed in [17], we believe that code generation is an important aspect of teaching modeling. The reason for this is that students receive automated feedback on their models, which helps them relate to their model using an artifact they are already familiar with, namely source code. This kind of feedback positively affects the learning outcomes, which could not be satisfactory achieved using a pen and paper.

The interesting question is then why the students perceive the tools negatively, especially for code generation purposes, even though we know from observations and from education literature that feedback supports learning. One explanation is that the tool exposes a lack of knowledge, or deficiencies in the model under development. If the students make mistakes in their models that they later use for code generation, the tool exposes this directly and immediately. In contrast, for imprecise models, there is much less feedback. Possibly, this leads to a feeling of knowledge' [42], a feeling that they understood the model, while they actually did not. Feeling of knowledge is often linked to situations where we think we can answer a question because it looks familiar or contains several familiar terms [42]. In the case of modeling, it might be that the students are familiar enough with the diagram types, e.g., through lectures, that they believe their solution is good. If this explanation is in fact valid, this would be a strong argument to use tools that provide automated feedback in modeling education, even if it results in negative perceptions.

Another possible explanation is the kind of feedback provided by modeling tools. In the case of Papyrus, the feedback students receive is often in the form of errors during design or code generation. However, we can observe that our students dislike this kind of feedback, an observation supported by Weaver [43] for written feedback. Similarly, several studies show that negative feedback in programming environments, e.g., compiler errors, heavily affect self-motivation and other important factors in students' learning, e.g., [44], [45], [46]. At the same time, motivation has been shown to have a significant effect on programming learning [47]. Hence, if the modeling tools affect motivation in a negative way due to the nature of feedback provided, this could have profound effects on learning. Therefore, it is valuable to investigate the use of more encouraging feedback customized particularly for modeling education, similar to approaches used in programming education, e.g., in [48]. While this can take the form of specialized modeling tools, it could also be related to the form of exercises performed by the students, e.g., using the extreme apprenticeship method proposed by Vihavainen et al. [49].

The nature of modeling feedback, or lack thereof, could explain why the perception of Umple in UoA 1 was uniquely positive (even when the UML data was not). Umple's feedback is based on errors found in the generated Java code. As students are familiar with Java errors and as Umple shows the errors usually in relation to the originating modeling elements, students are able to fix them with ease.

F. Summary

Summarizing the discussion, we can answer our three research questions as follows.

As discussed in related work on modeling education, modeling tools are often perceived as cumbersome by students (**RQ1**). However, our data shows that this perception is highly related to several factors. First, the tool support given in the course and how closely this support relates to the students' projects is of importance, as clearly seen in Case II. Secondly, it plays an important role whether or not the models created by students have to be formal, e.g., to allow for code generation. If formal models are required, this can lower the students' perceptions substantially. This suggests that when teaching modeling, there should be a balance between informal and formal models, without a focus solely on code generation. Finally, we observe that if formal models are used for code generation or other models transformations, the effectiveness of the resulting artifacts is important. If students clearly see that the process of transforming models into code supports their work even if this requires getting used to the process of doing so, it will positively affect their perception of the tools. If they instead feel like it is only an obstacle, they will perceive the tools as cumbersome and ultimately as a burden.

With respect to the students' perception of modeling, we observe that the course context the use of modeling tools plays a key role (**RQ2**). The use of informal models in Case II positively affected the students' perception of modeling, even to the extent that the perception of the modeling tool was secondary. In contrast, a strong focus on code generation in Case II seemingly affect the students' view of modeling negatively. Therefore, we need to distinguish in more detail what we mean by modeling education and to investigate in greater depth how we can support students and convey the material better in the different approaches.

Finally, there is a large overlap between modeling tool perception in industry and in education (**RQ3**). For example, usability is named in many studies that investigate modeling tools in industry, e.g., in [11], [5], as well as in modeling education, e.g., in [15], [16], [17], [18]. Similarly, it is a re-occuring topic in our own data. However, we also observe several differences in the perceptions of modeling tools. While characteristics such as interoperability [5], [12], [50], handling large models [13], or organisational aspects [51] play key roles in industry, this is not the case in education. For example, interoperability or handling large models play a minor role as students rarely work on projects complex enough to include multiple tools or modeling languages. In contrast, other tool characteristics are relevant. For example, we observed in our study the need for positive feedback to motivate students. Similarly, code generation and model transformation facilities might not need to be as powerful as for industrial use, but rather quick and effective for small-scale projects.

VI. Validity Threats

In the following, we discuss the threats to validity in our case study and countermeasures we took to reduce them. We follow the categorization by Runeson et al. into *External Validity*, *Internal Validity*, *Construct Validity* and *Reliability* [34].

A. External Validity

By design, case studies have a very limited external validity, stemming from a lack of control and from the fact that a topic is studied within its context. Therefore, we cannot claim that our findings extend beyond the studied cases. Instead, we try to describe the case context as detailed as possible, in order to allow readers to understand our environment and to decide whether or not the findings might generalize to their own education context.

B. Internal Validity

The students in our two cases come from a variety of different cultural backgrounds. This could influence the way they answer evaluation questionnaires. For example, the low hierarchy in Scandinavian society could mean that these students are more likely to voice concerns or annoyances in anonymous questionnaires. Similarly, students from another background might be overly positive or negative with their assessment. We believe that the collection of qualitative data in addition to quantitative data helps us to lower this threat, as we could use the feedback to evaluate our research questions using concrete suggestions in contrast to a score only.

C. Construct Validity

To avoid bias, we used identical questionnaires within each case. Additionally, the questionnaires across the two cases are close to identical, with only minor differences in phrasing. For example, in the US case the students were asked how useful the respective tool was for the project, while in the Swedish case the students were asked how the tool affected their project. While this might affect the outcome, we believe that other factors, such as the background and the teachers, will likely have a much stronger impact on the outcome. Still, this threat cannot be ruled out entirely.

D. Reliability

As discussed in the internal and construct validity sections, the results are most likely to some extent dependent on the student samples (and their backgrounds) and the teachers giving the course. While we cannot influence the students' characteristics, we try to make the course content as transparent as possible in order to enable replication and reproduction of our study. In particular, the introduction lectures to Papyrus in the Swedish case are available as screen casts online[4]. Similarly, the questions we asked the students are discussed in Section III-B and thus transparent.

VII. Conclusion

In this paper, we characterize some of the challenges and opportunities in modeling education using a case study method with two cases, one course on MDA in the US and one course on MBE in Sweden. We focused on students' perceptions of tools and modeling in general, supported by both qualitative and qualitative data.

We observe that students' perception of modeling effectiveness is influenced by multiple factors. One key finding in this work that students' perception tends to be more negative when the tool provides negative feedback, such as compiler errors in the generated code. Therefore, modeling tools tailored for education should consider to include positive feedback mechanisms, such as proposing improvements. While it might also be possible to not provide any feedback, this could lead to a 'feeling of knowledge' instead of actual modeling knowledge. A second finding is that the students perception of modeling seems to be more positive when informal models for early and ambiguous tasks such as requirements analysis or elicitation are part of the course content. If the projects are mainly focusing on generating executable system artifacts, models must be precise and have unambiguous semantics in order to support generation. As such, underlying tools require details and precise models, and can usually expose immediate deficiencies in the models. While this often frustrates students and can significantly hamper their perception of the tool, it could significantly improve learning outcomes. As such, constructive education-tailored feedback becomes instrumental.

References

[1] J. Börstler, L. Kuzniarz, C. Alphonce, W. B. Sanders, and M. Smialek, "Teaching software modeling in computing curricula," in *Proceedings of the Final Reports on Innovation and Technology in Computer Science Education 2012 Working Groups*, 2012, pp. 39–50.

[2] W. J. Dzidek, E. Arisholm, and L. C. Briand, "A realistic empirical evaluation of the costs and benefits of uml in software maintenance," *IEEE Transactions on Software Engineering*, vol. 34, no. 3, pp. 407–432, May 2008.

[3] R. Soley *et al.*, "Model driven architecture," *OMG white paper*, vol. 308, no. 308, p. 5, 2000.

[4] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 2012.

[5] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software & Systems Modeling*, pp. 1–23, 2016.

[6] O. Badreddin, T. C. Lethbridge, and M. Elassar, "Modeling practices in open source software," in *Open Source Software: Quality Verification: 9th IFIP WG 2.13 International Conference, OSS 2013, Koper-Capodistria, Slovenia, June 25-28, 2013. Proceedings*, E. Petrinja, G. Succi, N. El Ioini, and A. Sillitti, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 127–139.

[7] T. Gorschek, E. Tempero, and L. Angelis, "On the use of software design models in software development practice: An empirical investigation," *Journal of Systems and Software*, vol. 95, pp. 176–193, 2014.

[8] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso, and G. Reggio, "Preliminary Findings from a Survey on the MD* State of the Practice," in *Proc. of 2011 International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Sept 2011, pp. 372–375.

[4]https://www.youtube.com/channel/UCefaOkE8GpvBQjPGUndL7zQ

[9] E. Stiller and C. LeBlanc, "Effective software engineering pedagogy," *J. Comput. Sci. Coll.*, vol. 17, no. 6, pp. 124–134, May 2002.

[10] O. B. Badreddin, A. Sturm, A. Hamou-Lhadj, T. Lethbridge, W. Dixon, and R. Simmons, "The effects of education on students' perception of modeling in software engineering." in *HuFaMo@ MoDELS*, 2015, pp. 39–46.

[11] P. Mohagheghi, W. Gilani, A. Stefanescu, M. Fernandez, B. Nordmoen, and M. Fritzsche, "Where does model-driven engineering help? experiences from three industrial cases," *Software & Systems Modeling*, vol. 12, no. 3, pp. 619 – 639, July 2013.

[12] S. Kirstan and J. Zimmermann, "Evaluating costs and benefits of model-based development of embedded software systems in the car industry–results of a qualitative case study," in *Proc. of Workshop "From code centric to model centric: Evaluating the effectiveness of MDD (C2M:EEMDD)"*, 2010.

[13] P. Baker, S. Loh, and F. Weil, "Model-Driven Engineering in a Large Industrial Context – Motorola Case Study," in *Proc. of ACM/IEEE 8th International Conference On Model Driven Engineering Languages And Systems*. Springer Berlin Heidelberg, 2005.

[14] A. Forward, O. Badreddin, and T. C. Lethbridge, "Perceptions of software modeling: a survey of software practitioners," in *5th workshop from code centric to model centric: evaluating the effectiveness of MDD (C2M: EEMDD)*, 2010.

[15] T. Lethbridge, G. Mussbacher, A. Forward, and O. Badreddin, "Teaching UML using umple: Applying model-oriented programming in the classroom," in *Proc. of 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE T)*, May 2011, pp. 421–428.

[16] S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens, and D. R. Stikkolorum, "Tool use in software modelling education," in *ACM/IEEE 16th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, vol. 1134. CEUR-WS.org, 2013.

[17] G. Liebel, R. Heldal, and J. P. Steghfer, "Impact of the use of industrial modelling tools on modelling education," in *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, 2016, pp. 18–27.

[18] R. F. Paige, F. A. Polack, D. S. Kolovos, L. M. Rose, N. Matragkas, and J. R. Williams, "Bad modelling teaching practices," in *ACM/IEEE 17th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, 2014.

[19] E. Ramollari and D. Dranidis, "Studentuml: An educational tool supporting object-oriented analysis and design," *Proc. of 11th Panhellenic Conference on Informatics*, pp. 363–373, 2007.

[20] "ArgoUML," last accessed March 2017. [Online]. Available: http://argouml.tigris.org

[21] "UMLet, a Free UML tool for fast UML Diagrams," last accessed March 2017. [Online]. Available: http://www.umlet.com

[22] J. Cabot and D. S. Kolovos, "Human factors in the adoption of model-driven engineering: An educator's perspective," in *Advances in Conceptual Modeling: ER 2016 Workshops, AHA, MoBiD, MORE-BI, MReBA, QMMQ, SCME, and WM2SP, Gifu, Japan, November 14–17, 2016, Proceedings*, S. Link and J. C. Trujillo, Eds., 2016, pp. 207–217.

[23] I. Hammouda, H. Burden, R. Heldal, and M. R. V. Chaudron, "Case tools versus pencil and paper," in *ACM/IEEE 17th Int. Conf. on Model Driven Engineering Languages and Systems – Educators Symposium*, 2014.

[24] P. Stevens, "Updating the software engineering curriculum at edinburgh university," in *Proc. Software Engineering Education Symposium SEES*, vol. 98, pp. 188–193.

[25] E. Crahen, C. Alphonce, and P. Ventura, "Quickuml: A beginner's uml tool," in *Companion of 17th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. ACM, 2002, pp. 62–63.

[26] S. A. Turner, M. A. Pérez-Quiñones, and S. H. Edwards, "minimuml: A minimalist approach to uml diagramming for early computer science education," *J. Educ. Resour. Comput.*, vol. 5, no. 4, Dec. 2005.

[27] "Violet," last accessed March 2017. [Online]. Available: http://alexdp.free.fr/violetumleditor

[28] "Dia Diagram Editor," last accessed March 2017. [Online]. Available: http://dia-installer.de/shapes/UML/index.html.en

[29] K. M. Hansen and A. V. Ratzer, "Tool support for collaborative teaching and learning of object-oriented modeling," *SIGCSE Bull.*, vol. 34, no. 3, pp. 146–150, Jun. 2002.

[30] D. Batory and M. Azanza, "Teaching model-driven engineering from a relational database perspective," *Software & Systems Modeling*, pp. 1–25, 2015.

[31] T. Harris, "YUML," last accessed March 2017. [Online]. Available: http://yuml.me

[32] G. Dévai, G. F. Kovács, and Á. An, "Textual, executable, translatable uml." in *OCL@ MoDELS*, 2014, pp. 3–12.

[33] D. R. Stikkolorum, T. Ho-Quang, and M. R. V. Chaudron, "Revealing students' uml class diagram modelling strategies with webuml and logviz," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, 2015, pp. 275–279.

[34] P. Runeson, M. Höst, A. Rainer, and B. Regnell, *Case Study Research in Software Engineering*. Wiley Blackwell, 2012.

[35] R. K. Yin, *Case study: design and methods*, 4th ed., ser. Applied social research methods series. Sage, 2009.

[36] S. Gérard, C. Dumoulin, P. Tessier, and B. Selic, *19 Papyrus: A UML2 Tool for Domain-Specific Language Modeling*, 2010, pp. 361–368.

[37] J. Biggs, "Enhancing teaching through constructive alignment," *Higher Education*, vol. 32, no. 3, pp. 347–364, 1996.

[38] O. Badreddin, "Thematic review and analysis of grounded theory application in software engineering," *Advances in Software Engineering*, vol. 2013, p. 4, 2013.

[39] D. Steinberg, F. Budinsky, E. Merks, and M. Paternostro, *EMF: eclipse modeling framework*. Pearson Education, 2008.

[40] L. Kuzniarz and J. ürgen Börstler, "Teaching modeling: An initial classification of related issues," in *Proceedings of the 7th Educators Symposium @ MODELS 2011*, vol. 52, 2011, pp. 1–10.

[41] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The Annals of Mathematical Statistics*, vol. 18, no. 1, pp. 50–60, 1947.

[42] L. M. Reder and F. E. Ritter, "What determines initial feeling of knowing? familiarity with question terms, not with the answer." *Journal of Experimental Psychology: Learning, memory, and cognition*, vol. 18, no. 3, p. 435, 1992.

[43] M. R. Weaver, "Do students value feedback? student perceptions of tutors written responses," *Assessment & Evaluation in Higher Education*, vol. 31, no. 3, pp. 379–394, 2006.

[44] P. Kinnunen and B. Simon, "Experiencing programming assignments in cs1: the emotional toll," in *Proceedings of the Sixth international workshop on Computing education research*, 2010, pp. 77–86.

[45] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*, 2004, pp. 199–206.

[46] A. J. Ko, "Attitudes and self-efficacy in young adults' computing autobiographies," in *Visual Languages and Human-Centric Computing, 2009. VL/HCC 2009. IEEE Symposium on*. IEEE, 2009, pp. 67–74.

[47] S. Bergin and R. Reilly, "The influence of motivation and comfort-level on learning to program," in *Proceedings of the PPIG*, vol. 17, 2005, pp. 293–304.

[48] M. J. Lee and A. J. Ko, "Personifying programming tool feedback improves novice programmers' learning," in *Proceedings of the seventh international workshop on Computing education research*, 2011, pp. 109–116.

[49] A. Vihavainen, M. Paksula, and M. Luukkainen, "Extreme apprenticeship method in teaching programming for beginners," in *Proceedings of the 42nd ACM technical symposium on Computer science education*, 2011, pp. 93–98.

[50] P. Mohagheghi and V. Dehlen, "Where Is the Proof? – A Review of Experiences from Applying MDE in Industry," in *Proc. of 4th European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA)*. Springer Berlin Heidelberg, 2008.

[51] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden, and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" in *Proc. of ACM/IEEE 16th International Conference on Model Driven Engineering Languages and Systems*. Springer Berlin Heidelberg, 2013.