Thesis for The Degree of Doctor of Philosophy

Systematic Guidelines for Software Modelling: An Empirical Study on Enhancing Modelling Education and Training

Shalini Chakraborty

Department of Computer Science
Reykjavik University
Reykjavik, Iceland 2024

Systematic Guidelines for Software Modelling: An Empirical Study on Enhancing Modelling Education and Training

Shalini Chakraborty

Printed by x
Reykjavik, Iceland 2024.

"Today is a good day to try"
- The Hunchback of Notre Dame

Systematic Guidelines for Software Modelling: An Empirical Study on Enhancing Modelling Education and Training

Shalini Chakraborty

Abstract

Background: Software modelling holds significant promise for enhancing various aspects of software and systems engineering, including productivity and cost efficiency. Despite these advantages, its widespread adoption across the entire field remains limited. Extensive research has explored the reasons behind this limited adoption, uncovering issues such as subpar code generation, inadequate tool support, and a lack of guidance or training. Specifically, it has been suggested that engineers are reluctant to embrace modelling because it requires excessive effort and offers little usefulness, a view shaped by their educational background.

Aim: Overall, our goal is to conduct an empirical investigation with university students, understanding their perception of software modelling during their university studies. We aim to explore students' challenges with modelling assignments, tools, and the modelling content taught in courses. Additionally, we want to understand which aspects of modelling students find beneficial for learning and carry with them into their future academic or industry careers. Finally, we aim to create systematic guidelines for software modelling to be used by both students and instructors.

Method: To achieve our goal, we conducted several empirical studies with university students, teaching assistants, and instructors. We collected data through interviews, surveys, and observation studies. To evaluate the effectiveness of the systematic guidelines, we applied them to university courses where modelling was taught.

Results: The results described in the thesis are twofold: first, we present university students' perceptions of modelling, and then we describe the guidelines we created based on that perception. Our results show that students recognise the benefits of modelling, such as using models for planning and group communication, but their understanding is hindered by unclear assignment expectations, irregular and insufficient feedback, and lack of experience with problem domains. Finally, we conclude our thesis with systematic guidelines that will help students enhance their modelling skills and knowledge, guiding them to apply this knowledge in real-world industry settings.

Conclusion: Our results can potentially enhance education and training in software modelling, benefiting both academic settings and industrial environments. The modelling guidelines encourage students and instructors to follow

a structured approach, starting from understanding a modelling problem to selecting a suitable modelling strategy based on the problem domain and tools and ultimately interpreting the resulting model. The guidelines also assist instructors in providing regular and systematic feedback on students' modelling efforts. The guidelines improve communication between students and the course by clearly outlining expectations and values for modelling assignments, which reflects the value of modelling in different future endeavours.

Keywords

Software Engineering, Empirical Research, Software Modelling, Model-Based Engineering, Modelling Education

List of Publications

Appended publications

This thesis is based on the following publications:

[**Paper I**] **Shalini Chakraborty**, Grischa Liebel, *Modelling guidance in software engineering: a systematic literature review*
Software and Systems Modeling (2023), 1-17.

[**Paper II**] **Shalini Chakraborty**, Grischa Liebel, *We do not understand what it says–studying student perceptions of software modelling*
Empirical Software Engineering 28 (2023), 149.

[**Paper III**] **Shalini Chakraborty**, Javier Troya, Lola Burgueño, Grischa Liebel, *Exploring Actions, Interactions and Challenges in Software Modelling Tasks: An Empirical Investigation with Students*
Under review in Empirical Software Engineering.

[**Paper IV**] **Shalini Chakraborty**, Grischa Liebel, *Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling*
Empirical Software Engineering and Measurement, ESEM'2024.

Other publications

The following publications were published during my PhD studies, or are currently in submission/under revision. However, they are not appended to this thesis, due to contents overlapping that of appended publications or contents not related to the thesis.

[**a**] A. Ragnarsson, **S. Chakraborty** and G. Liebel, *ModRec: a tool to support empirical study design for Papyrus and the Eclipse Modeling Framework*
2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (2021), 361-369.

[**b**] G. Liebel and **S. Chakraborty**, *Ethical issues in empirical studies using student subjects: Re-visiting practices and perceptions*
Empirical Software Engineering 26 (2021), 1-37.

[**c**] G. Liebel, J. Klünder, R. Hebig, C. Lazik, I. Nunes, I. Graßl, J. Steghöfer, J. Exelmans, J. Oertel, K. Marquardt, K. Juhnke, K. Schneider, L. Gren, L. Happe, M. Herrmann, M. Wyrich, M. Tichy, M. Goulão, R. Wohlrab, R. Kalantari, R. Heinrich, S. Greiner, S. Rukmono, **S. Chakraborty**, S. Abrahão and V. Amaral, *Human Factors in Model-Driven Engineering: Future Research Goals and Initiatives for MDE*
Software and Systems Modeling 23 (4), 801-819.

Acknowledgment

The last five years of my life have been successful because I have had enormous support from the people around me and far away from me!

Thank you, Elli and Josh, for encouraging me to go for hikes and waiting patiently on the trails when I really questioned my choices! Sveinny and Luke, what a pleasure and fun to serve the RUMPS board with you two! I will cherish our board game nights together. Abdullah, Cari, Xavier and Kamal, thank you for being my amazing travel buddies! Thank you, Duncan, for always being a kind friend to me! I will miss our breakfast, elevenses and luncheons together at university. Raphaël, thank you for all the dinners together, conversations and all the Belgian chocolates you gave me. Naizeth, Vasiliki, Ilham, Jasmine and Majd, my ladies, it was an honour to work alongside you all. Thank you for all the support you gave me and all the memes we shared between us! Ioana, thank you for being there for me. I will miss our walks and lunches together. Lupita, thank you for inspiring me through your work and giving me company to our many retreats. Róbert, I'll miss our long conversations. Thank you for helping me through my teaching. Living far away from home is not easy; thank you, Satyaki, Tanushri Di and Avik Da, for being there when I missed home and home-cooked food. Thank you Loku, for sharing your Indian snacks with me.

I couldn't survive Iceland without a few "not so-fun" people. Albert, Embla, and Maxime, thank you for being weird, funny and amazing friends. I'll miss our talks and laughs!

Emil, I started my PhD by interviewing you. Thank you for finding me again after a year and staying with me. Thanks for your hugs, dad jokes, spaghetti bolognese, and being there whenever I needed you.

Ethan, thank you for supporting me during my thesis writing phase. You were always kind, lifting me when I was feeling low, and your constant encouragement made it easier to handle the challenges of work. You were proud of me on the days I really doubted myself. Love you!

I couldn't even think about doing a PhD in Iceland if my family hadn't been so supportive of me. Mani, baba, thank you for always being there and tolerating all my nonsense. I love you. Di, you have always inspired me to be a strong woman, excellent researcher and brave daughter. I couldn't imagine living alone if you weren't always a phone call away.

Thanks to Eric Knauss and Daniel Mendez for inviting me to Chalmers and

BTH, respectively, and providing me with great hospitality. Javier and Loli, thank you for your guidance and support in the last year of my PhD! Thank you to Reykjavik University, the Computer Science department and my research centre, CRESS, for providing such a supportive and welcoming environment. To all my research collaborators, thank you for guiding me through my PhD.

Finally, the person without whom I wouldn't be writing this thesis so proudly, my supervisor, Grischa. Thank you for supervising and inspiring me. You taught me everything: research, teaching, organising events and managing all that while writing papers. You are the best teacher in life I could have ever wanted. Danke!

Contents

Abstract iii

List of Publications v

Acknowledgement vii

I Summary 1

1 Introduction 3
1.1 Background . . . 5
1.1.1 Model-Based Engineering and Software Modelling . . . 5
1.1.2 Models in Industry . . . 5
1.1.3 Modelling in Education . . . 6
1.2 Research Goals and Research Questions . . . 8
1.2.1 G1: To understand the existing guidance of modelling in academia . . . 8
1.2.2 G2: To explore students' perception of modelling and to propose recommendations for modelling education . . . 10
1.2.3 G3: To evaluate the recommendations and create systematic guidelines for students to learn modelling efficiently and carry that knowledge into their future careers . . . 11
1.3 Research Methodology . . . 11

2 Summary of Included Papers and their Contribution to the Thesis 15
2.1 Paper I: Modelling Guidance in Software Engineering: A Systematic Literature Review . . . 15
2.2 Paper II: We do not understand what it says–studying student perceptions of software modelling . . . 18
2.3 Paper III: Exploring Actions, Interactions and Challenges in Software Modelling Tasks: An Empirical Investigation with Students . . . 20
2.4 Paper IV: Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling . . . 22

3 Discussion **25**
3.1 Literature on Modelling 25
3.2 Students' Perception of Modelling 26
3.3 What do Students' Model and How? 29
3.4 Modelling Guidelines for Education and Training 30
3.4.1 Iterative modelling and feedback including model interpretation 30
3.4.2 Limitation of diagram types 31
3.4.3 Grading Rubrics 32
3.4.4 Use of Familiar Problem Domain: 32
3.4.5 Modelling Tool 32
3.5 From university to industry: *How do we carry the knowledge?* 33

4 Validity Threats **37**
4.1 Internal Validity 37
4.2 External Validity 38
4.3 Conclusion Validity 38
4.4 Construct Validity 38
4.5 Reliability 39

5 Conclusion and Future Work **41**

Bibliography **43**

II Appended Papers **53**

6 Paper I: Modelling guidance in software engineering: a systematic literature review **55**

7 Paper II: We do not understand what it says–studying student perceptions of software modelling **73**

8 Paper III: Exploring Actions, Interactions and Challenges in Software Modelling Tasks: An Empirical Investigation with Students **101**

9 Paper IV: Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling **133**

Part I

Summary

Chapter 1

Introduction

Software modelling is a necessary practice in dealing with the increasing complexity of contemporary software systems, e.g., in robotics [1, p. 239] or cyber-physical systems [2, p. 157-158]. However, despite its potential benefits, the adoption of modelling in Software Engineering (SE) has been low, with important issues being insufficient tool support, organisational resistance, and a lack of guidance/training [3]–[10]. While technical issues have received considerable attention in the modelling community, there is a lack of work on guidance/training [3]–[6], [9], [10] and methodological guidelines are missing [1], [2], [11]. For instance, Schätz et al. state that "methodical guidelines are missing how to use suitable abstractions of (parts of) a cyber-physical systems at varying level of detail to enable the engineering of those systems with a sufficient level of confidence concerning the quality of the implemented systems" [11].

To improve guidance and methodological support for modelling, it is crucial to understand modelling education, how students learn modelling, and what they perceive from that knowledge. Further, understanding individuals' experiences, challenges, and preferences in modelling can significantly enhance the overall effectiveness of modelling guidance. Incorporating students' opinions on software modelling is vital, as it provides insights into their learning processes and helps identify areas where guidance is lacking. Software modelling is a creative task [12] in which abstraction plays a key role [13], and individual differences can strongly affect how models are created. In the related area of Business Process Modelling (BPM), existing work finds that individuals exhibit different *styles* when creating business process models [14]–[18]. However, we believe that the concept of software modelling style for individuals is still premature in SE, especially in an educational context where students often create models inconsistently to complete assignments. Therefore, we take into account the modelling preferences of students, which are in the premature state of modelling styles. While these preferences are not yet fully defined, they cannot be ignored as they have the potential to reveal the diverse ways students create models and highlight different approaches that should be considered when providing guidance. By addressing these individual preferences in our

modelling guidance, we aim to enhance the overall effectiveness of modelling education. Understanding and integrating students' opinions and experiences will allow us to tailor the education to their specific needs, ultimately leading to the development of more proficient and confident modellers. Additionally, while working in the industry, the scope of modelling becomes ambiguous. Most of the time the extent of modelling used in a project largely depends on the individual's knowledge, creativity, and the purpose of the model. These decisions, which ultimately determine the entire modelling application, are based on an individual's prior learning and experience with modelling. University education serves as the initial step in modelling experience, establishing a strong knowledge base.

Therefore, in the thesis, we investigate students' experiences with modelling at university and creating guidelines for modelling education. The thesis aims to **investigate the educational practices of software modelling in academia and create systematic guidelines to facilitate efficient learning and long-term application of modelling**. We break down the overall aim into three consecutive goals:

- G1: To understand the existing guidance of modelling in academia.
- G2: To explore students' perception of modelling and to propose recommendations for modelling education.
- G3: To evaluate the recommendations and create systematic guidelines for students to learn modelling efficiently and carry that knowledge into their future careers.

Goals G1 and G2 form the foundation of the thesis, supported by empirical evidence gathered from several studies. G1 aims to identify the current state of modelling in academia, focusing on existing guidance or training for software model creation. G2 explores students' perceptions of software modelling, their challenges with modelling and modelling education, and their methods of creating models. The aim is to develop potential guidelines for software modelling, which we refer to as recommendations in the paper. Finally, G3 aims to evaluate these recommendations produced in G2 and establish structured and systematic software modelling guidelines based on this evaluation and further empirical data. The guidelines from G3 will enhance the teaching and learning of software modelling, ensuring students gain a comprehensive understanding and can effectively apply their knowledge in both academic and industry settings.

The rest of this chapter is structured as follows: In Section 1.1, we discuss the background of our research problem, software modelling regarding MBE and software development, the related work on modelling in industry and education, and the importance of taking students' perceptions into account. Then, in Section 1.2, we outline the research questions (RQs) followed in our paper and align them with the three goals of the thesis. Finally, we conclude this chapter by describing the research methodology we used for the thesis in Section 1.3.

1.1 Background

In this section, we give an overview of the background of the research behind the PhD project.

1.1.1 Model-Based Engineering and Software Modelling

Model-Driven Engineering (MDE) or Model-based Engineering (MBE) is the current state-of-the-art in software abstraction, where models are used as primary artifacts in the SE process. These approaches have gained popularity in academia and industry for managing the complexity of modern software and are considered a significant advancement in software development [19]. Although the benefits of MBE are often considered obvious, higher abstraction levels do not guarantee better software and while code generation may boost productivity, the effort to develop models and make manual modifications can negate this benefit [20].

Regardless of the success of Model-Based Engineering (MBE), abstract models are crucial to the future of software development. In software modelling, UML and UML-based development methods have become the de facto standards in SE. Textbooks on UML and object-oriented design commonly refer to heuristics for creating UML diagrams, such as those by Rumbaugh, Bruegge, and Abbott [21]–[23].

Abbott's method [23], which involves identifying system objects by searching for nouns in informal descriptions, is a well-known example. While these heuristics are widely used for certain diagram types, like class diagrams, they are less common for behavioural diagrams. Moreover, there is no empirical evidence confirming the effectiveness of these heuristics.

In software modelling, several factors play a crucial role, including how a model is interpreted [24]–[28], the purpose behind creating the model—such as for communication [29] or for analysing interactions and dependencies among complex systems through interactive storytelling [30], [31] and the various aspects of modelling like using UML or sketching on different platforms, whether modelling tools or whiteboards [29]. Software modelling serves numerous purposes and can be approached in various ways, making it unrealistic to expect a straightforward, one-size-fits-all process. However, this very diversity underscores the necessity for clear and comprehensive guidance.

1.1.2 Models in Industry

There is substantial work on the adoption and the challenges of software modelling in industry [9], [10], [19], [20], [32]–[37].

In an early study at Motorola, Baker et al. [37] find that models increase productivity and reduce defects. However, they also find that modelling tools are insufficient and lack interoperability, and that MBE does not scale sufficiently.

Mohagheghi et al. [35], [36] highlight the potential of using models for simulation and testing, while also mentioning tool issues and model complexity

as challenges of adopting MBE.

Hutchinson et al. [19], [20], [32], [33] conduct several studies assessing the state of practice of MBE (Model Based Engineering) in industry. The authors confirm that modelling tools and model complexity are problematic, but also highlight organisational factors. Among others, they find that a lack of training hinders the adoption of modelling.

In a survey of the Italian software development industry, Torchiano et al. [34] found that developers primarily use models for informal sketches and communication. They also reported that inexperience among developers limits the use of models.

In a survey among systems engineering professionals, Liebel et al. [10] discovered that models provide substantial benefits, such as increased productivity. However, several challenges in both technical and non-technical areas remain, including a lack of training and guidance, tool shortcomings, and a lack of tool interoperability. Following up with an in-depth qualitative study at two automotive companies, Liebel et al. [9] reported that models are used primarily for communication and to handle complexity. Stakeholders also preferred sketches and informal models.

System models are often used in other domains, such as healthcare, for communication and to provide better organisational support. In [38], authors discuss the application of event-based models such as Petri nets to capture the various operations running in a hospital. Although found helpful, the authors also highlight certain challenges like needing more understanding in adopting the models from healthcare associations. Considering UML specifically, the healthcare domain has multiple evidence of using UML models to design the domains [39], [40] and human-centric tasks [41]. However, in the latter, the authors mentioned a need for more explicit representation from the models. In the healthcare domain where the cognitive behaviour of the actors is highly involved in the process, software models that are used for designing those processes should provide the ability to include the behaviours.

1.1.3 Modelling in Education

Research on models in education exists primarily in three forms: (i) solution proposals on how to teach modelling, including tools tailored to an educational context, (ii) experience reports on modelling education, and (iii) surveys among students.

An example of the first category is the practical approach to teaching model-driven software development proposed by Schmidt et al. [42]. In the proposed course, students develop a code generator using standard software development tools. Similarly, Westphal [43] describes the design of an SE course that focuses heavily on modelling. In [44], authors proposed a course where they used students both as language designers and its users to evaluate the usability of software language engineering (SLE). Gonnord et al. [45] try a reverse approach, and with the students, they journey from low-level C code to designing a modelling language workbench. In all cases, evaluation of the proposed course design relies on student feedback.

Numerous experience reports exist related to modelling education [46]–[48]. Akayama et al. [46] share their experience and opinion on tool use in software modelling education. The paper describes different approaches taken by the individual authors and provides a discussion of factors such as modelling tools vs pen and paper, the conflict between the concepts of design and programming, and how to measure the quality of models. Paige et al. [47] discuss what they consider to be bad practices of teaching modelling. They name bad practices such as covering a too broad range of modelling-related topics, and focusing on syntax instead of semantics. Similarly, Kolovos and Cabot [48] present a corpus of use cases for courses teaching MBE. The authors state that modelling courses regularly suffer from the use of uninteresting or irrelevant examples, and therefore propose a list of use cases suited to teach modelling. Moody [49] mentions a need for more effort in designing visual syntax for notations. Therefore, he proposes a tutorial that defines a "set of principles for designing cognitively effective visual notations: ones that are optimised for human communication and problem solving". Ciccozzi et al. [50] present a survey among educators on how modelling is taught. The authors find that educators see the focus on tools critical, as it might prevent students from understanding the core principles of modelling.

Several empirical studies exist on modelling education[51]–[55]. Reuter et al. [51] study students' problems with UML diagrams over two modelling courses. As a result, the authors present a catalogue of problems with UML diagrams. In a similar direction, Stikkolorum et al. [52] study student problems and modelling strategies in UML Class diagrams by presenting students with a specialised UML editor that incorporates feedback mechanisms. The results reveal four distinct strategies (depth-less, depth-first, breadth-first and ad hoc) which are four different ways to draw a class diagram for a given problem. These strategies are based on how they draw the class diagram with necessary associations and attributes in the given task and a number of problems, such as choosing the right syntax elements. Agner et al. [53] conduct a survey on modelling tool use among 117 students. The authors report issues such as a lack of feedback, difficulties in drawing the diagrams and tool complexity. Hammouda et al. [55] compare the use of modelling tools to pen and paper use. Using a survey, they evaluate students' perceptions of the differences, finding no clear advantage for either approach. In the context of modelling tools in education, the tool Umple needs to be highlighted [56]. Umple is a tool that allows to create UML models in a textual concrete syntax close to object-oriented languages like Java. Furthermore, code in many different general-purpose programmming languages can generated directly from Umple models, thus allowing for direct feedback from models. Surveys with students have shown positive results on the use of Umple [54], [57]. Umple is successfully used by several instructors teaching modelling. However, to our knowledge, it is currently not widely used on a global scale.

Liebel et al. [54], [58] conducted several case studies on tool use in modelling education. Tools like Umple and Papyrus were used in the case studies.

The authors find that students can use industrial modelling tools successfully, but require substantial coaching in how to use the tools, with a dedicated

tool champion present in the course. They further report that the tools' inability to provide adequate feedback impacts the tool acceptance. In an attempt to connect industry practice to education, Whittle and Hutchinson [59] present essential differences between industry practice and education concerning software modelling. The authors find that modelling education is more UML-centric. In contrast, the industry is placing more emphasis on abstract models, that is, working at the meta modelling level with in-house domain-specific languages and code generators. Furthermore, in industry, it is more common to use a bottom-up approach to adoption, whereas modelling is typically taught top-down. In the bottom-up approach, developers try different modelling aspects to refactor existing modules and not think about the whole system abstraction unlike the top-down approach, where the typical development starts with modelling the system requirements.

In summary, while the importance of software modelling is recognised, many challenges regarding its adoption remain. Technical challenges have received attention, but guidance has largely been unexplored. Heuristics for model creation exist in software engineering, but they are not empirically validated. Although related research, such as on the understandability of models, exists in software engineering, there is a lack of empirical studies on how to create models. However, studies in business process modelling (BPM) follow similar aims and methods to our intended work, providing guidance and demonstrating the feasibility and potential value of such research.

1.2 Research Goals and Research Questions

The PhD research is conducted in three phases, each phase fulfilling one goal. One or more RQs were designed for each goal and carried out specific studies to address them, which resulted in several papers. The thesis ended in four papers, addressing six RQs and achieving three goals. Figure 1.1 shows the outline of the thesis. On the left, we mention three phases, the goals of each phase and the number of papers as the outcome of each phase. On the right, describe the RQs dedicated to each goal. Table 1.1 shows the papers and the RQs answered in each paper.

The following subsections detail the three goals, the RQs, and the studies conducted to address these questions.

1.2.1 G1: To understand the existing guidance of modelling in academia

The PhD project started by investigating the existing guidelines on software modelling in literature. G1 was dedicated to better understanding the current state, especially shortcomings in guidance and training in modelling. For this purpose, the following RQ was created:

- RQ1: What kind of guidance exists in Software Engineering (SE) literature on model creation?

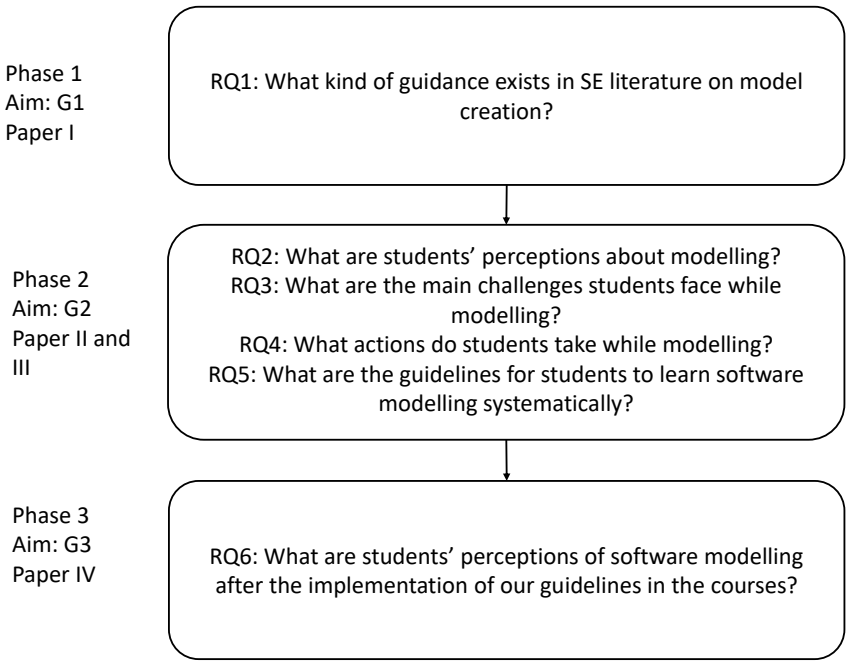


Figure 1.1: Outline of the thesis: Goals, RQs and Papers

Papers	Research Questions
Paper I: Modelling guidance in software engineering: a systematic literature review	RQ1
Paper II: We do not understand what it says–studying student perceptions of software modelling	RQ2, RQ3 and RQ5
Paper III: Exploring Student Actions, Interactions and Challenges in Modeling Tasks: An Observational Study	RQ3, RQ4 and RQ5
Paper IV: Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling	RQ6

Table 1.1: Papers included in the thesis and the answered RQs in each paper

RQ1 was essential for the thesis, aimed at creating systematic guidelines for widely used processes in software modelling. It was crucial to gather all existing state-of-the-art information, particularly because MBE is not yet a widely adopted approach. However, software modelling, and specifically UML modelling, has been used for a while and is a regular subject in most universitys'

SE courses. The gap between informal modelling and the application of software modelling for MBE necessitated this investigation due to the lack of guidance for software modellers. The term "lack of guidance" is vague, and before providing appropriate guidance and training, we needed to examine the empirical data available in the literature. While studies like Pinggera's [60], [61] focus heavily on modelling style, the standardised terms associated with modelling were not clear. This uncertainty prompted us to create RQ1 and perform a systematic literature review.

The outcome of the systematic literature review is Paper I, which highlighted the fragmented nature of existing research and the inconsistent use of terminology. By defining key terms such as modelling *guidelines*, *methods*, and *styles*, we aim to aid clarity in future research. This work contributes significantly to our goal of understanding existing guidance in academia and establishes the foundation for future studies to develop comprehensive guidelines for software modellers.

1.2.2 G2: To explore students' perception of modelling and to propose recommendations for modelling education

Phase 1 resulted in the finding that there is a lack of guidance in literature and an inconsistent use of modelling terminologies. In addition to reviewing the past SE literature, it was necessary to understand the existing modelling practice in education. It was also essential to see if the condition in the literature matches the university education in terms of guiding individuals in modelling. Therefore, to analyse the results from phase 1 and make better systematic modelling guidelines, we started phase 2 to explore students' modelling perceptions, challenges, and styles and to come up with our potential set of guidelines. Four RQs were used to achieve G2.

- RQ2: What are students' perceptions about modelling?
- RQ3: What are the main challenges students face while modelling?
- RQ4: What actions do students take while modelling?
- RQ5: What are the guidelines for students to learn software modelling systematically?

Phase 2 strengthens the empirical base for our guidelines, with four RQs dedicated to collecting valuable data from academia. RQ2 explored students' overall experiences with software modelling, focusing exclusively on UML as it is the standard in university courses. This question aimed to understand students' perceptions of modelling, based on the provided materials in different courses. RQ3 investigated the challenges of modelling, which are crucial for understanding students' perception of it. RQ3 helped to better reason with the answers from RQ2. RQ4 investigated into individual modelling styles, especially how an individual creates a model, building on a key finding from phase 1 that

cognitive abilities must be considered. RQ5 combined all the answers from RQ2-RQ4 and aimed to produce our first set of guidelines towards a systematic software modelling practice and learning. Papers II and III were outcomes of phase 2, fulfilling G2. Paper II involved interviews with students from three universities to explore RQ2 and RQ3, while Paper III used observations with students from two universities to answer RQ3 and RQ4. Although RQ3 was common to both papers, the methodologies differed: interviews in Paper II and observations in Paper III involving different student populations to understand their modelling challenges and actions. Both papers outlined *recommendations* as a potential guideline set evaluated in phase 3.

1.2.3 G3: To evaluate the recommendations and create systematic guidelines for students to learn modelling efficiently and carry that knowledge into their future careers

Finally, in phase 3, we used phases 1 and 2 results and evaluated our modelling guidelines by applying them to university courses. And finally, we conclude our PhD project with our evaluated set of systematic modelling guidelines. We followed one RQ for this phase:

- RQ6: What are students' perceptions of software modelling after the implementation of our guidelines in the courses?

Analysing the results from RQ2, RQ3, and RQ4, we drafted our initial modelling guidelines at the end of phase 2, which we refer to as *recommendations*. These recommendations are based on academic data gathered from multiple case studies with university students and professors/modelling instructors across three countries and five universities. The recommendations address three key aspects of modelling: communication, planning, and documentation, and training modellers to create appropriate models. We proposed these recommendations in Papers II and III. After completing RQ5, evaluating the efficiency of our guidelines became necessary. Thus, to answer RQ6, we conducted an evaluation study in Paper IV. In this study, we applied the recommendations in two university courses, analysed students' feedback through assignment submissions, surveys, and instructor interviews, and conducted a focus group with an education expert, a modelling expert, and the modelling instructor from both courses. Following this evaluation, we refined our recommendations and concluded G3 and our PhD project by producing four systematic modelling guidelines. Figure 1.2 shows the intermediate outcomes and final conclusion after three phases. It also demonstrates the data flow in between phases.

The final guidelines are discussed in Section 3.

1.3 Research Methodology

Empirical research has become increasingly crucial in the field of software engineering, particularly in understanding the complexities and challenges

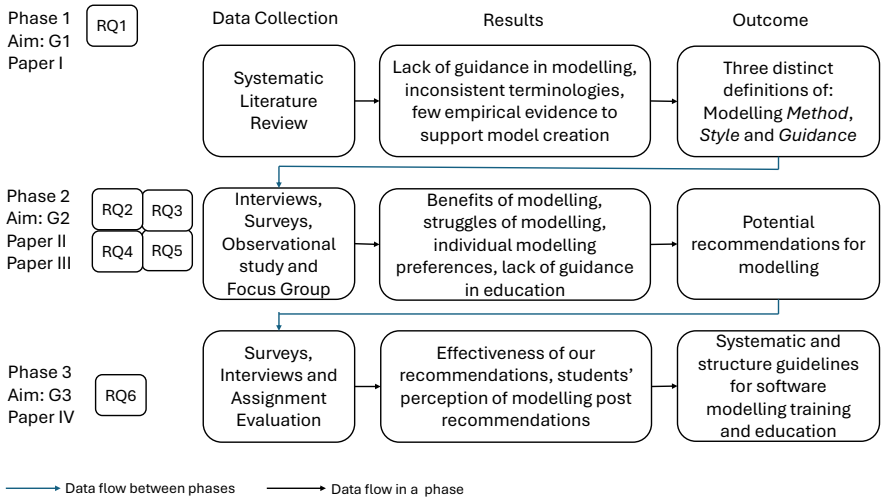


Figure 1.2: Data Flow and Results from the three Phases

associated with software development processes [62], [63]. The evidence-based paradigm has gained significant recognition in various domains, resulting in research outcomes that have a greater influence on practitioners and policy-makers. This approach is now being explored in the field of software engineering [64]. Empirical evidence is crucial as it provides a solid foundation for making informed decisions based on observed and measured phenomena. In academia and professional practice, relying on empirical data enhances the credibility and reliability of findings, leading to more effective and applicable solutions. By grounding theories and practices in empirical evidence, we ensure that conclusions are not merely speculative but are supported by real-world data. This approach not only increases the integrity of research but also ensures that results and observations are both practical and beneficial. Qualitative research methods, such as case studies, have proven invaluable in this regard, providing researchers and practitioners with a deeper, more nuanced understanding of the real-world issues and constraints faced by software development teams [65], [66].

We embarked on an empirical research journey of five years to achieve our overall thesis goal. In this timeline, we performed several studies with outcomes in four papers. The methodologies used in each paper are described in Table 1.2

In Paper I, we conducted a systematic literature review (SLR), a secondary form of study used to identify and evaluate available research relevant to a certain research question or topic of interest [62]. We chose an SLR over a mapping study [67], as we were interested in the detailed picture, not the broad coverage of a research area a mapping study provides [68]. We selected an SLR as the first step of the thesis as it is critical in providing a comprehensive and unbiased summary of the available evidence, modelling guidance in our case. It

Papers	Research Methodology	Data Source
Paper I	Systematic Literature Review	Search 1: 8604 papers, Search 2: 3572 papers
Paper II	Case Study (Interviews)	16 students, 5 instructors
Paper III	Observational Study	32 students
Paper IV	Case Study (Interview, Survey, and Assignment Evaluation)	61 students participated in the survey, 85 students provided assignments for evaluation, 5 instructors interviewed

Table 1.2: Research Methodologies and Data Sources Used in the Papers

also helps avoid the selective reporting that can occur in narrative reviews [69]. Additionally, by following a predefined protocol, this evidence-based practice ensures transparency and reproducibility, allowing other researchers to assess the validity of the conclusions drawn.

For Papers II and III, our primary focus was on gathering empirical data to inform our guidelines. Both papers involved human subjects, with students and instructors serving as our key sources of information. Paper II adopted a multiple case study approach, where we conducted interviews with a total of 16 students and 5 instructors across three universities and five distinct courses. Alongside interviews, we analysed assignment details associated with these courses. Participation in the interviewees was voluntary and instructors were not aware of who participated in the interviews. The semi-structured interviews included a blend of *direct* and *open-ended* questions. *Direct* questions, such as asking about the tools students used in the course, were combined with *open-ended* questions like, "For which purposes or in which situations in software engineering do you think modelling is useful?" Each interview commenced with a brief introduction to the research being conducted and was recorded with the participants' consent. The duration of the interviews was set, varying based on the participant group, with 40 minutes allocated for students and 30 minutes for instructors.

In Paper III, our research employed an observational study where students engaged in modelling tasks for an hour while we observed their actions and interactions.

This approach aimed to capture firsthand behaviours and interactions that might otherwise go unnoticed. While substantial information can be gathered through observation, techniques such as think-aloud protocols are essential for participants to verbalise their thought processes [63], [70]. Hence, we chose to observe the actions and interactions of participants to gain insights into their internal processes. A total of 32 students from two universities participated in the study. Students signed up in pairs, choosing their own partners. We recorded their screens and the communication between each pair during the hour-long modelling session.

For Paper IV, we engaged with university students to evaluate the efficiency, quality andd suitability of four modelling recommendations introduced during

their bachelor courses. For this paper, the methodology was a mix of survey with students, interviews with instructors, assessment of students' assignments and a focus group session. As the recommendations were in early stages of development, employing focus groups was efficient, as they provide an excellent opportunity to gather initial feedback [71]. The specific objective was to evaluate the recommendations within a defined context and explore participants' opinions and experiences regarding their use.

For all our studies, we carefully followed the ethical standards. All individuals signed a consent form before participation, where they were made aware of the research being carried out. Participation was voluntary, and participants could withdraw from the respective study at any time without penalty. Additionally, participants could deny the researchers the right to use the data collected from their participation.

We followed different data analysis methods through out different studies.

For Paper I, we performed paper selection in three rounds. In the beginning of each round, both authors went through a random 5 per cent of the remaining papers to determine inter-rater agreement. We used Fleiss kappa to measure our agreement, a statistical measure used to evaluate the reliability of an agreement between a fixed number of raters [72]. We decided on a threshold value of $\kappa > 0.7$ as a minimum agreement to continue the selection process. In Paper II, both authors performed in-vivo coding individually and discussed the outcomes for the first five interviews. After this initial stage, we jointly grouped the codes into themes based on the scope of that study and then performed thematic analysis and analysed the rest of the interviews. In Paper III, we collected video and audio recordings. All authors performed open coding on these data and classified them into appropriate themes. Similar to Paper II, we initially selected four pairs of data, performed the analysis, and discussed the results among the four authors. Once we reached an agreement, the first author completed the remaining data analysis. For data analysis in Paper IV, we used three approaches. Firstly, we conducted in-vivo coding to analyse feedback from both students and instructors. Secondly, we developed an analysis protocol, the Diagram Assessment Sheet (DAS), to thoroughly examine all assignment submissions, including the diagrams drawn by the students. Finally, for the focus group, both researchers used open code analysis.

Details of the methodology conducted in each of these papers can be found in **Part II: Appended Papers**.

Chapter 2

Summary of Included Papers and their Contribution to the Thesis

2.1 Paper I: Modelling Guidance in Software Engineering: A Systematic Literature Review

As a first step towards providing the necessary guidance in modelling, we conducted a systematic literature review to explore the current state of the art. We searched academic literature for guidance on model creation and found that research on model creation guidance is fragmented, with inconsistent terminology and a lack of empirical validation or supporting evidence. In Paper I, we outline the different dimensions commonly used to provide guidance on software and system model creation. Additionally, we provide definitions for the three terms: modelling *method*, *style*, and *guideline*, as current literature lacks a well-defined distinction between them.

Problem

Despite the potential benefits of using models in Software Engineering (SE), adoption has been low, typically pointing to issues such as tool support, organisational resistance, and a lack of guidance/training [3]–[10]. Technical issues have historically received substantial attention in the modelling community, seen for instance by the large amount of tools for modelling and Model-Based Engineering (MBE). However, work on guiding individuals in creating models receives typically only marginal attention.

To better understand shortcomings in literature on guidance and training in modelling, this paper reports on a Systematic Literature Review (SLR) [62] surveying work on guidelines, styles, or training approaches on creating

software and system models in SE. We address the following research question **RQ: What kind of guidance exists in SE literature on model creation?**

Methodology

We conducted an SLR, a secondary form of study used to identify and evaluate available research relevant to a certain research question or topic of interest [62]. We chose an SLR over a mapping study, as we were interested in the detailed picture, not the broad coverage of a research area a mapping study provides [68]. We started the review by manually reading through the papers from last five years since the starting of this paper (2015-2019) appearing in *MODELS conference* (`http://modelsconference.org/`) and *SoSyM journal* (`https://link.springer.com/journal/volumesAndIssues/10270`), the two prime venues for SE modelling research. We conducted two reviews, S1 and S2, with the following search strings:

S1: **(“modeling” OR “modelling” OR “model-driven” OR “model-based”)**
AND (“guidelines” OR “training” OR “styles” OR “creation”)
AND (“software engineering”).

—

S2: **(“modeling” OR “modelling” OR “model-driven” OR “model-based”)**
AND (“approach” OR “process” OR “method” OR “template”)
AND (“software engineering”).

In S1, we put search terms like *guidelines* and *creation* that are directly related to our RQ and in S2, we put generic terms as *approach* and *method* to capture the broader aspects of software model creation. We applied both search strings to search on Scopus, IEEE Xplore, ACM Digital Library, and Web of Science, resulting in 8604 papers from S1 and 3572 from S2. We performed three rounds of selection for both sets of papers and an additional snowballing, resulting in a final set of 28 papers from S1 and S2 and 7 papers from the snowball search for analysis.

Contribution

Using two search strings (S1 and S2) and snowballing, we extracted 35 papers from 80,051 for full-text reading. The contribution of this paper is twofold: first, we found the current state-of-the-art situation of modelling discussed in the literature, including terminologies to be used inconsistently, lack of cognition explored in software modelling and little empirically validated modelling guidance. Second, to address inconsistent terminology, we define modelling *guidelines*, *methods*, and *styles*, aiding clarity in future research.

This paper significantly contributes to our first PhD project goal (G1: To understand the existing guidance on modelling in academia) and helps establish the foundations for future studies. Since we discovered a gap related

to cognition in software modelling, which is considered and proved important in business process modelling, we designed our future studies involving cognition and team dynamics. The three distinct definitions, especially modelling style, helped us understand how an individual models, shaping the guidelines that concluded the PhD project.

2.2 Paper II: We do not understand what it says–studying student perceptions of software modelling

In our second paper, we investigate the perceptions of students towards modelling in a university environment. We conducted a multiple case study with 5 cases (5 courses from 3 universities) and two units of analysis (student and instructor). We collected data through 21 semi-structured interviews, which we analysed using in-vivo coding and thematic analysis. Our findings help in understanding better why students struggle with software modelling, and might be reluctant to adopt it later on.

Problem

There is a lack of in-depth studies that aim to understand how students perceive modelling and why this is the case. Experience reports commonly suffer from various biases, as they rely on individual experiences and lack rigorous methods of data collection and analysis. Similarly, existing quantitative studies aim to provide a broad picture and can, therefore, not provide detailed explanations [73]. Finally, experience reports by researchers that have a particular focus on software and systems modelling risk being unrepresentative of instructors without such a focus. Since students will after graduation become professionals, it is important to understand their perceptions. Understanding existing challenges they face, but also perceived benefits of using models can help improve modelling education and facilitate an increased uptake in industry.

To address this gap, paper II aims to investigate students' perceptions of modelling, both in terms of benefits of and challenges with modelling. We pose the following two research questions (RQs) to address our aim:

- RQ1: What are students' perceptions about modelling?
- RQ2: What are the main challenges students face while modelling?

Methodology

We adopted a multiple case study approach, where we conducted interviews with a total of 16 students across three universities and five distinct courses. Additionally, instructors from these courses were interviewed, encompassing both bachelor's and master's level programmes. Alongside interviews, we analysed assignment details associated with these courses. Participation in the interviews was voluntary, and instructors were not aware of who participated in the interviews. Participants could withdraw their involvement at any time and deny the researchers the ability to use the data. All interviews were online. To appreciate students' participation, we offered them movie vouchers or did a donation to charity on their behalf.

After conducting the interviews, we transcribed the interviews using transcription services[1]. The two researchers then applied in-vivo coding separately on each interviewee transcript. In-vivo coding is one of the first cycle coding used for qualitative analysis [74], where literal spoken words are used as codes, instead of assigning own terms (open coding). This helps reducing the risk that interviewees' words are misrepresented at the coding stage. The total time of data collection and analysis was approximately one year.

Contribution

Paper II summarises the results that contributed significantly to achieving G2. A clear picture of students' perception of modelling reflected through the results, such as students recognising the benefits of modelling for planning and communication while often perceiving it as not worth the effort or a waste of time. They struggle with the fast pace of courses and lack of in-depth feedback beyond diagram syntax. We also identify differences from existing literature and provide deeper explanations for observed phenomena. For instance, our interviewees reported minimal tooling-related challenges, likely because they were allowed to choose their tools, and none of the courses required specific modelling tools. We find that students often struggle to understand modelling due to a concurrent lack of knowledge in related areas, such as the problem domain, the intended task (design or architecture), object orientation, and programming skills. Without automatic feedback from modelling tools, students cannot anchor their models to any known domain.

Based on the result we were able to recommend our first set of potential systematic guidelines for software modelling education, which were evaluated through the studies in phase 3. We reported four recommendations in Paper II. **Iterative and project-based learning** for modelling education so challenges like irregular feedback and unclear expectations can be minimised. Working with a project-based approach within a group will make students clearer about the purpose of the assignments and the diagrams used. Students will be able to embrace benefits like planning and maintaining better communication. We suggest **limiting the number of diagram types** used in a course, so students can make mistakes and not get overwhelmed by the pressure of assignments. By using **familiar problem domains** and giving students the freedom in notation and tooling, educators can help students understand modelling in their time and pace. Unlike coding, the instructors must provide students with constructive feedback, as modelling doesn't offer automated errors, at least not in the early stages when students are not generating code from the models. Hence, having **grading rubrics for assignments where stated model purposes and grading are aligned**, and aligning that with each task is very important for the students' understanding.

[1]We used Konch[2] and Go Transcript[3].

2.3 Paper III: Exploring Actions, Interactions and Challenges in Software Modelling Tasks: An Empirical Investigation with Students

Moving to Paper III, our research involved an observational study with 32 students from two different universities. To achieve G2, we needed to observe students creating models to understand individual styles in reading and solving modelling problems. In Paper II, we analysed students' experiences with modelling through interviews. Paper III builds on this by exploring students' interactions and actions during the modelling process.

Problem

Research on students' perspectives on modelling needs to provide more detailed explanations, such as specific challenges and benefits regarding modelling assignments, modelling tools, syntax, and semantics. In our Paper II, we engaged in 16 interviews with students and 5 interviews with instructors to explore students' perceived understanding of modelling and the challenges they encountered. In this paper, we aim to explore students' modelling knowledge and modelling actions. Further, we want to investigate students' challenges while solving a modelling task on specific modelling tools. The study follows two research questions (RQs):

- RQ1: What actions do students take on a modelling tool to solve a task?
- RQ2: What challenges do students face while solving modelling tasks?

Methodology

We conducted an empirical study by observing 32 students from two different universities, Reykjavik University, Iceland, and University of Malaga, Spain, solving modelling tasks for one hour.

Students signed up in pairs, selecting their own partners. Participation was voluntary, and participants could withdraw from the study at any time, without penalty. Similar to Paper II, participants could deny the researchers the right to use the data collected from their participation. For the study, students were given a modelling problem domain with three tasks, solvable by class and sequence diagrams, and were asked to follow an approach inspired by pair programming. For the first problem, student A acted as the driver, while student B was the navigator. This meant that student A solved the problem by drawing diagrams, while student B navigated through the problem, providing insights and inspecting the drawing. For the second problem, the students switched roles. Each pair got one laptop and a recorder to record their conversations, and we also asked them to use screen recording so we could record their modelling activities. In UMA, students used Magicdraw as they

were familiar with that tool. In RU, students used PlantUML as it was the tool used in that course. Students received the problem description in English.

Students had 5 minutes to read the problem description, 25 minutes to solve the first problem (class diagram), a break of 5 minutes and then 25 minutes to solve the second problem (sequence diagram). We collected three forms of data: (i) *modelling actions through screen recordings*, (ii) *voice recordings of the conversations between a pair*, and (iii) *PDFs of each pairs' final diagrams*.

Contribution

Combined with Paper II, Paper III fulfils G2 by demonstrating students' perceptions and struggles with software modelling. Paper III contributes two pivotal insights to our PhD project: first, it provides stronger empirical evidence of students' perceptions of modelling and the challenges they face, going beyond mere opinions and backed by observations that include students' struggles not only with creating models but also with interpreting modelling problems. Observing students creating models and analysing their conversations about solving modelling problems gave us reasoning and depth to the results reported in Paper II. The findings highlighted several issues, such as inconsistent strategies, a lack of training in interpreting modelling problems, and the absence of methods to verify one's final model. This information enabled us to provide a more systematic structure to our guidelines, reported and evaluated in Paper IV. We discuss our final guidelines set in Section 3.

Second, this study allowed us to observe individual modelling preferences. Applying our definition of modelling style from Paper I, we identified several modelling preferences among students. With more detailed and extensive investigation, these preferences could evolve into distinct modelling styles, which can be essential aspects of software modelling. Since the goal of our PhD is to create a systematic guideline for software modelling education, we believe that considering individual styles is essential, although it may come later in the process of learning software modelling. We discuss this further in Section 5.

2.4 Paper IV: Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling

Paper IV reports evaluating the effectiveness of our modelling recommendations proposed after phase 2. We applied the recommendations in two bachelor courses at Reykjavik University to discover students' perceptions of software modelling after taking a course with the recommendations included. We conducted a mixed-method study, including interviews with teaching assistants, student surveys, and a focus group study involving students, teaching assistants, and experts from both modelling and education.

Problem

In Paper II, we present four recommendations for modelling education in universities after an extensive case study based on three universities. The recommendations are to be applied to modelling courses or courses where modelling is a sub-part. The recommendations included assignment structures, grading instructions and feedback suggestions. In this paper, We aim to evaluate the four recommendations and examine students' perception of modelling after taking a course with the recommendations.

Therefore, to address this goal we have the following research questions (RQs):

- RQ1: How effective are the recommendations in improving students' software modelling skills?
- RQ2: What are students' perceptions of software modelling after the implementation of the recommendations in the courses?

Methodology

We incorporated the recommendations in two university courses, focusing on software design and emphasising UML modelling. One of the authors assumed the role of instructor in both instances. We integrated a combination of the suggested recommendations to optimise the learning experience. Both courses were part of the same university and designed for bachelor-level students.

For the data collection process, we gathered students' assignments along with their questions throughout the assignment phases. Post-course, we conducted a survey to gain insights into their overall experience. We interviewed teaching assistants (TAs) for their perspectives on the recommendations and overall student participation. Additionally, we conducted a focus group [71] session to evaluate the recommendations. Prior to recording the focus group study, we obtained consent from all participants. Moreover, students received a movie voucher for their participation in the focus group study. We used informed consent to gather all data. Students signed a consent form, consenting to grant

access to their assignments and grading information for this study. The post course survey was voluntary for students. We also obtained informed consents from TAs for the interviews. In both surveys and interviews, participants had the right to withdraw at any moment, and they also retained the right to withdraw their consent at any time.

Contribution

Paper IV fulfils G3 and concludes pur thesis. The data gathered in the evaluation study and presented in Paper IV reflects the effectiveness of our recommendations. Additionally, we were able to refine and enhance our recommendations with further information. We identified areas for improvement, particularly in guiding students to apply their knowledge in real-world industry settings. Consequently, we propose modifications to the recommendations, incorporating specific insights from our study to enhance students' modelling experience in university settings and ensure long-term effectiveness. The final set of guidelines are presented and discussed in the Section 3.

Chapter 3

Discussion

We observed several significant outcomes from the four papers that comprise this thesis. Beyond the development of systematic guidelines, the research conducted through this PhD project reveals major findings in modelling education, the lack of cognitive exploration in modelling, and students' overall experiences with modelling in universities. This includes their current perceptions and actions towards modelling. The systematic guidelines are built upon an understanding of students' current attitudes towards modelling and should be applied to university courses where both instructors and students actively participate in enhancing modelling training. Finally, we discuss how to extend this modelling training into their future careers.

Results from each paper are discussed later through **Chapters 6-9**. Below, we discuss the summarised observations and findings of the thesis.

3.1 Literature on Modelling

Through our literature review, we found numerous solution proposals but only one piece of empirically validating research. These statistics highlight a significant lack of empirical evaluation in the literature. In contrast, the papers from BPM show a higher level of evaluation and validation. This suggests that BPM is more advanced in supporting model creation through empirical studies. Similarly, the only paper explicitly considering cognition and the construction of mental models was in the BPM area.

This lack of empirical studies is also common in software modelling. Zhang et al. [75] find that empirical research methods within SE are mostly applied to software maintenance, quality and testing. While software models and methods gained empirical attention[1] but none of them investigating modelling style nor model creation.

We utilised two search strings: one with direct keywords (S1), such as "guidelines" and "creation," and another with generic terms (S2), such as "approach" and "method." From S1, we discovered various papers that, according

[1] 66 papers published from 2013-2017 at EMSE and ESEM.

to our definitions, propose guidelines and methods. However, these papers often use terms like “guidelines,” “methods,” and “frameworks” somewhat arbitrarily. To address this issue, we propose definitions that build on existing ones as much as possible and outline the dimensions and properties of existing modelling guidelines and methods.

Interestingly, the six papers retrieved through S2 all propose methods according to our definition, suggesting that the terminology is not entirely arbitrary, as guidelines were exclusively found in S1. This indicates that ”methods” might be perceived as more comprehensive by researchers, commonly including stepwise instructions. Notably, we did not find any papers in S1 or S2 that propose a modelling style according to our definition.

Common ways to provide guidance in modelling include specifying which diagrams should be created, prescribing views or perspectives to be considered, encouraging stepwise decomposition or refinement, and suggesting general best practices or anti-patterns. Apart from stepwise decomposition/refinement, these forms of guidance can either follow a specific set of steps or not. For example, modelling guidelines might simply list a few UML diagrams to be created in any order, whereas a modelling method could prescribe the same diagrams in a stepwise manner.

In BPM, substantial work explicitly discusses the role of cognitive processes and individual preferences in creating business process models. However, in the SE modelling community, we do not find similar work outside of the BPM domain. This disparity may be due to the relatively specialised nature of BPM compared to general system modelling.

Notations, levels of abstraction, and purposes vary considerably in software and systems modelling, likely affecting the cognitive processes and modelling styles of individuals. Nevertheless, it is relevant to explore this direction further in the future. Existing work from the BPM community can provide valuable guidance for designing similar studies in software modelling as a whole.

To achieve this, we need to incorporate expertise from other fields, such as psychology, to better understand and support the cognitive processes involved in software modelling. This interdisciplinary approach could lead to a deeper understanding of individual modelling styles and the development of more effective modelling techniques and education.

3.2 Students’ Perception of Modelling

Students see specific benefits of modelling, primarily for areas where informal models might be sufficient, such as obtaining a system overview or conceptual understanding of the problem domain. However, many are skeptical as to what value modelling has for detailed system design.

We identify four categories where modelling benefits students: *Being on the Same Page, Better Planning, Better Understanding of Code,* and *Maintenance/Documentation.*

Many students find that modelling enhances group communication, helping them to express ideas, share work, and make collective decisions, thereby

keeping everyone on the same page. Students also find modelling useful for planning the development and designing systems. They appreciate being able to visually map the system and plan the development process through modelling. Regarding programming, some students find models helpful for obtaining a higher-level overview of the code. One student noted, *"I really like the idea of seeing it as a language to talk about code, because I will never read someone's code if they ask for feedback."* This is linked to a top-down modelling approach, where a better understanding of the system is achieved by breaking it down into steps. As one interviewee mentioned, *"If you are making a program that has more than 200 lines of code, then you probably need to model it."* Students also see modelling as beneficial for future tasks such as maintaining and documenting the product. However, only students with industry backgrounds mentioned this benefit in our interviews. From our analysis of assignments and interview data, we found that none of the courses explicitly mention maintenance/documentation. Assignments require an overview of the diagrams but do not include requirements for documentation. The importance of maintaining documentation and the methods for doing so still need to be clarified for students.

Despite benefits observed by the majority of the students, they experience several challenges related to modelling. These relate, among others, to *choose the right notation*, *what to express in the models*, and *how to apply it to unfamiliar domains.*

In university modelling courses, students typically receive assignments to create models. However, they often struggle to understand what is expected of them, and how to create the models. In particular, this is caused by a lack of knowledge in programming, software architecture, and other practical skills necessary to envision a "good" design. Additionally, assignments are often formulated in a way that is in contrast to how they are assessed, e.g., by asking students to create a *prescriptive* model, which is then assessed by how closely it resembles the final code, i.e., in a *descriptive* way. Students quickly pick up this discrepancy in grading and optimise their efforts towards the grading. That is, they take shortcuts initially, and simply update the model later to reflect their actual code. The evaluation criteria often demand *"Diagrams are well structured and provide good overview"* and *"Explanation of good design suggestions"*[2] which is vague and do not leave space for constructive, detailed feedbacks. As modelling is often a qualitative task, teachers often revert to giving feedback on objective things such as the diagram syntax. However, this is not perceived as useful feedback, as a student mentioned, "So no one can say anything to us. It's good or not. Just we should follow some rules about the diagram, for example, what is solid line, what is dashed line. Instead, students would like more "holding hands" with regular feedback. In particular, since there is no automated feedback as, e.g., in programming, students require timely and regular feedback.

Deep knowledge of the problem domain is necessary to perform many software engineering tasks in a satisfactory manner, e.g., programming [76] or

[2]Both these criteria are used precisely as it is in the two courses that we considered for the interview study

program understanding [77]. If this domain knowledge is lacking or entirely missing, software engineering tasks cannot be performed well. Allowing students to work with a known problem increases their interest and participation. Several students stated that, in addition to just learning modelling, they were also lacking domain knowledge and programming experience. Therefore, they were unable to relate their models to familiar concepts. Compared to other introductory topics, such as programming, this makes modelling particularly complex: Students are expected to learn a new concept (modelling), neither having problem domain nor software design/architecture experience. In addition, the models used in the courses we studied do not provide any kind of automated feedback, such as compiler/runtime error messages in programming. This leads to feelings of "just drawing something" in students, without an anchor to connect their models to. In a similar direction, the chosen problem domain might also affect how representative the learning is for actual systems in industry, or how suited modelling is for the given problem. For instance, one student noted: *"we got to know all the basics and the whole idea and how it's supposed to look like, but not how to use it in the future on the bigger picture and the scale of the app"*

Students complain that modelling takes time, and courses are tightly scheduled. Students feel pressure due to the bulk of assignments, especially when they are not fully aware of what is expected from them. Given that pressure, they try to prioritise and minimise effort where possible. However, arguably, learning how to model will require the students to make many mistakes and correct them in iterations. Together with the lack of feedback mentioned above, students perceive these iterative improvements as a waste of time.

Interestingly, these students did not perceive that iterative changes helped them understand the problem domain better, but rather saw it as a burden with unclear benefit. One of the reasons is courses contain individual assignments without a continuation or connection between themselves. For each problem students are instructed to draw a specific diagram and then move on to something else.

Students find it hard to remember the syntax of different diagrams and their purpose, pointing to a lack of prior knowledge of UML. One instructor offered the explanation that this relates to a lack of experience with object-oriented programming in general. *"they don't have a lot of experience using object-oriented programming."* This quote reinforces our discussion above, that students are exposed to various unknown activities when learning how to model, and therefore lack knowledge they can anchor their modelling experiences to. Students did not voice any significant tool challenges, other than their dislike for the tooling interface. Students used multiple tools for their assignments, e.g., Draw.io, PlantUML, and Lucidchart. Also, some students used plain pen and paper for drawing and communicating their diagrams. While offering choice in tooling clearly addresses many tool issues, it has the limitation that it only works if the course and the course assignments do not rely on specific tool features, such as code generation or simulation capabilities. This is a limitation in our study, as all our cases introduced UML only as a means to document, plan, and to communicate, without any automated processing of models in the

form of model transformation or other facilities.

Several of the challenges perceived by students might not be caused by software modelling as a course topic, but rather by the pedagogical quality of the courses. We did not try to assess the quality of the individual courses, and given the chosen research method we cannot control for it, either. However, we cover a variety of expertise in modelling on the lecturers' side, ranging from instructors that do not work on models in research and do not have a dedicated interest in modelling, to instructors that publish in software modelling venues. Therefore, we do not believe that the students' experiences can be related to issues in teaching only. The challenges students expressed are likely triggered by a combination of individual teaching styles, the course syllabus and the placement of the course in the overall program, the types of assignments and their assessment, and student preferences. As such, we believe it is rarely possible to claim that challenges arise due to poor teaching.

3.3 What do Students' Model and How?

We examined both students' completed assignments and their experiences during the course. Additionally, we conducted a separate study observing students modelling in pairs. Two key aspects to discuss here are students' understanding of what they are modelling and their modelling actions or preferences.

Giving students multiple assignments with different diagram types is useful, but stating a clear purpose is necessary for them to understand *what* they are modelling. Student assignments show that students need help understanding the correctness of a diagram. Students emphasise syntax, which is concerning as most of the modelling experts argue the importance of semantics over syntax. Two factors triggered this aspect among students. First, students perceive correctness primarily as syntactical correctness. Second, the concept of semantics remained unclear to them. However, students need help understanding a diagram's semantics regarding a problem description, as one of them mentioned, *The difficulties I have encountered is learning how to understand the analysis of the diagram and try to figure out if you have gather sufficient information.*

Students often struggle with *unclear expectations* in modelling courses [78]. This arises firstly from students encountering modelling diagrams for the first time and secondly from the overwhelming nature of UML. Learning and practising diagrams within a limited time frame can only provide limited assistance to students in mastering modelling. Since the syntax of UML is complex, their learning tends to be limited to these concerns, not moving beyond mastery of syntax.

When it comes to modelling, students use several strategies based on the modelling tool, the way they read the problem, and their personal style. For example, we observed two main preferences for connecting different classes in a class diagram. The first preference involves drawing all classes initially and establishing connections later, while the second preference entails sequentially

drawing classes and their associations. Students using PlantUML[3], a text-based modelling tool, follow the first preference. In contrast, students using MagicDraw[4], a more visually-oriented tool, favour the second preference. This suggests that the tool significantly influences these choices. Consequently, individual thinking influenced certain preferences among the students using different tools. In PlantUML, copying and pasting was a common trend, and during brainstorming, we observed that students often paused their actions. In contrast, students using MagicDraw frequently utilised many edit and delete actions, playing around with the interface by dragging parts of the diagram, especially in the sequence diagrams. Students use inconsistent strategies while solving the modelling problem. One reason behind this inconsistency is how students interpret the modelling problem. Throughout our studies, we noted that there is no systematic method taught to students for practising the reading of a modelling problem, which is a significant issue. The consequence of this challenge is that students' understanding of the correct diagram can be misleading.

In conclusion, students are unclear about what they are modelling, as the courses do not include a systematic method for teaching and assessing modelling. There's a lack of guidance in interpreting a modelling problem or applying modelling knowledge learned in universities to later careers. Students exhibit hints of modelling style, which we refer to in our paper as modelling preferences, as these are still premature and require further development. While the implications of these modelling preferences are promising and could be useful for students in their later careers, the preferences are currently disorganised and fragmented.

3.4 Modelling Guidelines for Education and Training

The PhD research concluded with eight guidelines for modelling education and training. These guidelines are established based on data collected throughout the PhD project from four different universities (Reykjavik University, Chalmers, Blekinge Institute of technology, University of Malaga) in three different countries (Iceland, Sweden and Spain).

3.4.1 Iterative modelling and feedback including model interpretation

Among the challenges observed from students, the two most voiced challenges are *Unclear Expectations* and *Irregular Feedback*. **Iterative, project-based learning (PBL)** [79] has the biggest potential to address these challenges. Within an iterative environment, students can understand modelling at their own pace, particularly in a first-year bachelor's course where they are introduced

[3]https://plantuml.com/
[4]https://www.magicdraw.com/

to UML for the first time. Iterative and consistent feedback fosters communication between students and instructors, enhancing students' understanding of modelling.

It is essential for students to engage in more reading of diagrams, not just of diagrams created by other student groups, but also of diagrams produced by professionals. It is important to discuss strengths and completeness of a diagram given a problem description, and to teach student to explore what makes a good diagram. During a focus group study in phase 3, the modelling expert mentioned, *"If I show you a class diagram for code generation, it will look ridiculous because it's so detailed, because it needs to be very precise and technical. Whereas, you know, I've seen company diagrams on their architecture, which were literally four boxes with a bunch of random arrows between them"*

Therefore, we propose that, in addition to iterative projects, we should include diagram interpretation as part of the modelling assignment. This would serve as an essential precursor, allowing students to grasp important concepts before jumping into diagram creation themselves. By starting with interpretation, students can develop a deeper understanding of the purpose and significance of each diagram component. Furthermore, it enhances communication between instructors and students regarding feedback, establishing clear expectations not only for assignments but also for students' participation in any modelling activities. Hence, we propose the following three guidelines:

- **G1 Structure modelling courses around iterative projects**
- **G2 Emphasis the importance of model interpretation as the initial step before commencing modelling activities**
- **G3 Provide ongoing and consistent feedback for each stage of the modelling process to facilitate student learning and growth**

3.4.2 Limitation of diagram types

UML constitutes a vast and intricate domain, presenting a substantial challenge for learners. To enhance the learning experience, it is essential to advocate for a more realistic approach that acknowledges the limitations of students' time and cognitive capacity. Hence, we propose narrowing down the array of diagram types covered in the course. By being more realistic about what students can feasibly learn within a limited time frame, we aim to prevent overwhelming them with an extensive variety of diagrams. We encounter the challenge of "unclear expectations," where students struggle to align a given problem with the appropriate modelling diagram. The essential thing here is to show students why those particular diagram types are selected. Previous research shows the connectivity between class diagram and object-oriented programming [80]. Similarly, we observe that students can see the connection between class diagrams and object-oriented programming concepts, helping them to bridge new concepts to existing knowledge in programming. Further, we encourage instructors to anchor diagram learning with more theoretical or technical concepts. For example, a student mentioned, *"I thought the sequence diagram was an interesting way to look at users interacting with the system."*

Therefore, the fourth guideline is:

- **G4 Limit the number of modelling diagrams covered in the syllabus to avoid overwhelming students and provide a rationale for the selection of each modelling diagram type, explaining why they are relevant and beneficial for the course objectives**

3.4.3 Grading Rubrics

Essentially, modelling is a tool for understanding a domain and planning or designing a system. In industry, these diagrams help comprehend problems and plan solutions. In a course, however, students learn syntactical aspects first, before learning their use. The grading rubric could focus initially on technicality and completeness, then progress to evaluating the effectiveness of the diagrams in addressing a specified purpose. This shift from syntax to purpose is crucial for preparing students for real-world applications. Also, rubrics can be a great tool of communication between TAs, TAs and students and between students themselves. We suggest to use rubric more into that direction. Therefore, in addition to a 'technical rubric', we propose to add an evaluation rubric for students to assess their modelling knowledge:

- **G5 Develop a technical grading rubric for assignments, focusing on evaluating the correctness and completeness of students' modelling diagrams**
- **G6 Introduce a second rubric for students to assess the effectiveness of the learned diagrams. Include checklists in this rubric to enable students to evaluate their learning against the course's learning outcomes**

3.4.4 Use of Familiar Problem Domain:

We advocate for instructors adopting a unified approach by selecting a single domain for the entire project, one that students are relatively familiar with. This approach not only fosters student engagement but also allows instructors to tailor the project to align with students' interests and the objectives of the course. Furthermore, it allows students to spend time on the concepts that are introduced in the respective course, namely software modelling, without too much cognitive load spent for understanding a new domain.

- **G7 Ensure that all assignments in the course revolve around a single problem domain. Encourage instructors to involve students in selecting the problem domain by highlighting the importance of diverse domains and their unique information and development requirements**

3.4.5 Modelling Tool

One of the critical revelations from our results is that students' approaches to modelling vary based on the tools they use. Modelling tools are often

considered challenging by students [81], [82]. However, beyond this challenge, modelling tools also shape students' modelling actions. Studies like [83]–[85] have investigated and explored different ways a tool can assist modellers; however, these are industry-level studies and do not focus on training students in modelling education. Additionally, the selection of a modelling tool is often left to the students. The issue lies in expecting students to enter the industry, use a modelling tool they have likely never fully utilised, and produce good products. It is beneficial to let students choose their own modelling tools, but we strongly recommend demonstrating the interfaces of various tools and allowing students to explore them through different assignments. Modelling tools can bridge the gap between academic and industry practices, shapes students' attitudes towards modelling, and enables them to manage its complexities more effectively. Hence, our final guideline is:

- **G8 Demonstrate different modelling tools in the classroom and assign tasks that encourage students to explore these tools**

Our guidelines aim not only to train students in modelling but also to help them gain a comprehensive understanding of it. While the guidelines may seem straightforward and akin to general educational guidelines, they are specifically designed to enhance communication between students, instructors, and the modelling process. These guidelines build on the benefits students have already experienced with modelling and address the issues they have expressed. Importantly, they clarify the purpose of modelling for students, ensuring that their attitudes towards modelling stem from sufficient and effective training.

3.5 From university to industry: *How do we carry the knowledge?*

While the PhD project focused purely on the educational context, it is interesting to discuss potential similarities and differences to industrial practice. First, we observe that models are appreciated for communication purposes and to handle system complexity in our cases, something that is also reported in industry [10], [86], [87]. Other benefits reported in industry, such as simulation or verification capabilities do not apply in our cases, since all five courses used modelling only on an informal level to express models for planning, communication, and coordination. Similarly, it is hard to reason about improvements reported from industry, e.g., in terms of productivity [35], [37].

Tooling issues are a frequent topic in industry, e.g., reported in [9], [10], [19], [20], [33], [88]. In contrast, only a few students raised tool issues. One explanation for this observation is clearly that none of our courses mandated a dedicated modelling tool for modelling, but instead left that choice to the students. Similarly, while models were often assessed for semantic correctness, the specified purposes of the models did not require syntactically or semantically correct models. Together with a lack of requirements for interoperability between tools, this removes most of the issues contemporary modelling tools have. Nevertheless, it is positive to observe that our student participants did

not finish their courses on modelling with the perception that modelling tools are bad, as is commonly reported in literature on modelling education, e.g., in [46], [58]. Such a negative perception could lead to a reduced uptake of modelling in industry.

Our interviewees raise several challenges that relate to a lack of guidance, feedback, and clarity when it comes to modelling. While they primarily perceive this as an issue in how assignments are set up and how the courses are designed, the challenges resonate with those in industry. For instance, a lack of training and guidance is raised in several empirical studies on modelling in industry, e.g., [10], [34], [86]. Whittle et al. [88] highlight that organisational and process factors play a major role in the use of MDE. Similar issues are raised by our interviewees in the educational context, namely that modelling assignments need suitable processes that allow for iterations and quick feedback loops. This is especially important as models were not used for automated tasks in any of our cases, i.e., there was no automatic feedback generated.

An important difference of the educational setting to industry is that students often lack both the technical background, e.g., in terms of programming experience, and the domain knowledge required for the example domains. Therefore, they struggle to contextualise the value and the quality of their models, often leading to a perception of doing a meaningless drawing task. Publications on modelling education sometimes argue for the use of realistic examples and caution against toy examples, e.g., [48]. However, also realistic examples have their pitfalls. While unrealistic domains are just that, unrealistic, they can also expose students to a known domain, or at least a low level of complexity due to a domain that is artificial and potentially more controlled and restricted. From this point of view, toy examples can be a suitable tool for modelling education. Given our findings, we advise to use primarily example domains the students have sufficient knowledge of, as discussed in the guidelines.

Students across different studies expressed their long-term opinions on modelling. Some held positive views, such as, *"I find that all of the diagrams we have learned to make have helped me gain a better understanding of what it takes to create software and are great for learning everything for the future."* Others had concerns, asking, *"If you look beyond the course, if you would apply this somewhere else now, either in a course or in industry, do you feel like it would still be something that would help you to roughly know what you should do?"* While we are not able to answer this based on our study, we would like to highlight the possibility that these opinions are shaped during university education and then maintained later on.

The primary challenge identified from our data is guiding students in the right direction and ensuring they retain their modelling knowledge for their future careers. Whittle and Hutchinson [89] find various mismatches between industry and educational modelling practice. The study reveals that, while modelling in education is very UML-centric, industry increasingly prioritises other notations, particularly at the meta-modelling level with bespoke domain-specific languages and code generators. In their study, Akundi et al. [90] conducted a two-phase study to check to which extent model-based systems engineering (MBSE) academic curricula in the US reflect industry workforce

hiring requirements. The study provided a pathway to align university curricula with industry needs and strategies to mitigate adoption challenges. While UML can serve as a tool for entry-level courses to modelling, eventually education might have to bridge to industry practice. This applies to handling various tools and notations, as mentioned above, but also to move beyond syntactical issues and use models purposefully to engineer software systems. Note that this requires a certain level of maturity and is therefore not suited for entry-level courses, where students have little knowledge of software engineering as a whole. However, over the course of an undergraduate degree, students should be able to gain an in-depth understanding and receive thorough training in modelling. That is, **addressing these issues while educating modellers** could lead to a higher uptake and appreciation of modelling in industry. Similarly, **making the use of models in industry more transparent** could be beneficial, also for purposes where formal models might not be required, such as documentation or communication.

Chapter 4

Validity Threats

In this section, we provide an overview of the threats to validity identified in this thesis research. Detailed threats and the mitigation strategies employed for each study are discussed in the respective papers included in Chapters 6-9 of this thesis. We adhered to the classification of threats as internal, external, conclusion, construct, and reliability as outlined in [91].

4.1 Internal Validity

Internal validity focuses on how confident we can be that the elements of the study actually caused the observed outcome [91]. There may be other factors influencing the outcome, which are beyond our control or have not been measured.

To mitigate this threat, we conducted pilot studies prior to the actual research to identify potential factors related to time, setup, and participants that could affect the outcome. Given that several interviews and surveys were conducted, as detailed in Papers II and IV, we consistently used open-ended questions to ensure participants could express their opinions honestly, without external influences or biases. For data analysis, we opted for in-vivo coding [92] to capture the direct opinions of our interviewees.

Despite these efforts, specific concerns for internal validity remained in the thesis. In Paper I, we had to reduce a large number of initial papers to a manageable amount for detailed analysis and dealt with ambiguous terminology. We excluded papers strongly in the first two exclusion rounds (based on title and venue, and based on abstract). This approach could lead to the exclusion of papers containing valuable guidance on model creation that were not clearly stated in their abstracts. Balancing search coverage and detailed analysis, we accepted this trade-off.

As reported in Papers II and III, our study participants were multilingual. Although everything was translated into English, there remained a possibility that participants might not fully express themselves appropriately in English.

4.2 External Validity

External validity is concerned with whether we can generalise the results beyond the scope of our study [91].

The majority of the studies conducted during this PhD research project are exploratory case studies [93], primarily using data collected through interviews. Our analysis leaned towards an interpretivist approach, which posits that *"humans construct knowledge as they interpret their experiences of and in the world; rejecting the objectivist notion that knowledge is simply there to be identified and collected"* [94], [95].

In interpretivist research, generalisability is different and rather difficult to achieve. As [96] states, *"one should not talk about a sample of cases, given that one would not aim to generalise to a population. Instead, one would like to generalise to similar contexts, and find supporting cases and conflicting cases for theories, and by doing that being able to conduct cross-case comparison."* Case studies should not be dismissed because they represent only a single case; each case contributes significantly to learning, particularly as they provide a deep understanding of a situation in a specific context [96], [97].

All our studies involve university students, either at the bachelor's or master's level. This increases the threat to external validity. Despite this, we included strong cases from four different universities across three countries, making the results valuable for understanding the study topic of modelling.

4.3 Conclusion Validity

Conclusion validity focuses on how confident we can be that the study elements and settings are genuinely related to the observed outcome and whether the study can be repeated [91].

In this research, multiple authors conducted the analysis, which introduces the potential for researcher bias. Although we implemented strategies to mitigate this, such as cross-checking and discussing findings among the team, some level of bias is inevitable. Additionally, the students who participated in our studies successfully completed their courses, meaning exact replication of the study with the same participants is not possible. While the study setting can be replicated, the population may differ, potentially affecting the outcomes. Furthermore, the dynamic nature of educational environments and the evolving curriculum could influence the consistency of the findings over time. Despite these challenges, we believe our conclusions offer valuable insights into modelling education, while acknowledging the inherent limitations in conclusion validity.

4.4 Construct Validity

Construct validity reflects to what extent the measures represent the construct investigated [91],

In our research, several threats to construct validity were identified, primarily stemming from the challenges in accurately capturing the constructs of interest, such as students' attitudes towards modelling and their modelling competencies. To mitigate these threats, we consistently employed pilot studies or preliminary studies prior to the main research. These pilot studies allowed for the refinement of instruments and procedures, ensuring they were effectively capturing the intended constructs. By testing and revising the measurement tools in these preliminary phases, we aimed to enhance the accuracy and reliability of the data collected in the main studies, thereby strengthening the overall construct validity of the research.

As reported in Paper I, we investigate model creation, guidance, and related topics. A threat to the validity of our study is that these topics are not described using established, standardised terminology in the modelling community. Therefore, it might in some cases be difficult to decide when a paper in fact provides guidance for model creation, and when not. To reduce this threat, we used open coding for data analysis, without relying on fixed keywords being used in the respective papers.

A potential threat stated in Paper IV, is that the four recommendations were applied in parts of two courses. As the courses have more to offer, the other parts (such as non-modelling theory/assignments) influence students' experience of the recommendations, which leads to the perception of modelling. To address this issue, we gathered students' opinions and real experiences through surveys and assignments. We also supplemented the data with the perspectives of teachers by interviewing TAs. Additionally, we held a focus group study to obtain more targeted feedback on the recommendations from students, TAs, and experts in modelling and education.

4.5 Reliability

Reliability in research refers to the consistency and dependability of the results obtained from a study. Ensuring reliability involves making certain that if the study were to be repeated under similar conditions by same or different researchers, the same results would be obtained [91].

In our research, we took several steps to enhance reliability and address potential threats. First of all, we published all data sets, interview guides either online or in the Appendix, so that the study can be replicated. In Paper I, we collected over 78,000 papers for the litearture review. The different steps of data collection and analysis are clearly described , providing a detailed roadmap for other researchers to follow and repeat the study. Nevertheless, there is an inherent subjectivity in several parts of our study, particularly in the exclusion and analysis steps of the review. To mitigate this, we aimed to be conservative in our exclusion criteria and calculated inter-rater agreement at all exclusion steps to ensure consistency and reduce bias. Despite these efforts, it is acknowledged that complete objectivity is challenging to achieve, but these measures were taken to enhance the reliability and robustness of our findings. In Paper II, we interviewed 16 students and 5 instructors. We published the

interviews for which we received consent. However, we could not publish some interviews as we did not obtain consent for disclosure.

Chapter 5

Conclusion and Future Work

The PhD thesis makes significant contributions to the education of software modelling. The research was centred on modelling education and training, with data collected and analysed from university environments, involving students, instructors, and teaching assistants. The primary goal was to understand the current state of guidance provided in modelling education and, based on this understanding and empirical evidence, develop a systematic set of guidelines to help students learn and apply modelling more efficiently, thus enhancing the overall training for modelling.

In [98], the author explains, *"There is no essential reason why models are required for building a software system. The system itself consists of code that provides the required functionality. A complexivist mindset acknowledges the effort invested into modeling, but, in turn, draws attention to the effort that is save from the use of models, in the context of realistic development settings: Communication problems, misunderstandings and design flaws can be discovered early, before the system is built, which can help avoid significantly greater effort and time for fixing problems after the system has been built. To support development of this mindset, educational activities should be designed in a way that makes it likely for students to encounter situations that allow them to appreciate the advantages of modeling, e.g., in terms of saved effort."*

Through six RQs and four papers, the PhD thesis provides the necessary guidance which was lacking in modelling education. For researchers, our results highlight significant gaps, such as the need for empirical studies on cognition in software modelling. For practitioners, our results demonstrate the various forms modelling guidance can take, providing inspiration on how engineers can be supported in creating and maintaining models for different purposes, such as formal analyses, increased understandability, or communication among stakeholders. We recommend iterative and project-based learning for modelling education to address challenges like irregular feedback and unclear expectations. Group-based projects can help students understand the purpose of assignments and diagrams, improving communication and planning skills. Limiting the

number of diagram types in a course can reduce overwhelm and allow for deeper learning. Using familiar problem domains and allowing freedom in notation and tooling can help students learn at their own pace. These guidelines align with existing beliefs and highlight new details for effective modelling education.

The next step in our research is to translate these guidelines into industry practice. To accomplish this, we intend to conduct further studies that explore into the practical application of modelling within professional settings. This will involve adapting our guidelines to better fit industry training requirements, ensuring that they remain relevant and effective for real-world scenarios. The transition from academic to industrial application is crucial, as it will allow us to refine our guidelines based on practical feedback and enhance their applicability in diverse professional environments.

In addition to investigating students' perceptions of modelling, we have identified the potential for exploring various modelling styles among them. These preliminary modelling preferences, discussed in Paper III, are in their early stages and require more empirical data to be firmly established as recognised styles. Future research should aim to include a broader and more varied sample of students, employing different tools and addressing a diverse array of problem domains. This approach will help validate and refine these emerging styles, contributing valuable insights into modelling practices.

Moreover, our modelling guidelines significantly foster communication between students, instructors, and the modelling concepts introduced in the course. Effective communication is frequently cited by students as a major benefit of modelling, and reinforcing this aspect could further enhance their learning experience. Future research should therefore focus on understanding how modelling impacts communication and vice versa. By exploring this dynamic interaction, we can strengthen the educational framework and better support students in mastering modelling techniques.

Bibliography

[1] Robotics, SPARC, "Robotics 2020 multi-annual roadmap for robotics in europe," *SPARC Robotics, EU-Robotics AISBL, The Hauge, The Netherlands*, vol. 5, 2016 (cit. on p. 3).

[2] E. Geisberger and M. Broy, *agendaCPS: Integrierte Forschungsagenda Cyber-Physical Systems*. Springer, 2012, vol. 1 (cit. on p. 3).

[3] J. Hutchinson, M. Rouncefield and J. Whittle, "Model-driven engineering practices in industry," in *33rd International Conference on Software Engineering (ICSE '11)*, 2011, pp. 633–642 (cit. on pp. 3, 15).

[4] J. Hutchinson, J. Whittle, M. Rouncefield and S. Kristoffersen, "Empirical assessment of MDE in industry," in *33rd International Conference on Software Engineering (ICSE '11)*, 2011, pp. 471–480 (cit. on pp. 3, 15).

[5] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" In *Model-Driven Engineering Languages and Systems*, ser. Lecture Notes in Computer Science, A. Moreira, B. Schätz, J. Gray, A. Vallecillo and P. Clarke, Eds., vol. 8107, Springer Berlin Heidelberg, 2013, pp. 1–17 (cit. on pp. 3, 15).

[6] J. Hutchinson, J. Whittle and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming*, vol. 89, Part B, no. 0, pp. 144–161, 2014, SI: Success Stories in Model Driven Engineering (cit. on pp. 3, 15).

[7] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen and M. Fritzsche, "Where does model-driven engineering help? experiences from three industrial cases," *Software and Systems Modeling*, vol. 12, no. 3, pp. 619–639, 2013 (cit. on pp. 3, 15).

[8] P. Mohagheghi and V. Dehlen, "Where is the proof? - a review of experiences from applying mde in industry," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, I. Schieferdecker and A. Hartman, Eds., vol. 5095, Springer Berlin Heidelberg, 2008, pp. 432–443 (cit. on pp. 3, 15).

[9] G. Liebel, M. Tichy and E. Knauss, "Use, potential, and showstoppers of models in automotive requirements engineering," *Software & Systems Modeling*, 2018. DOI: 10.1007/s10270-018-0683-4 (cit. on pp. 3, 5, 6, 15, 33).

[10] 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*, vol. 17, no. 1, pp. 91–113, 2018. DOI: 10.1007/s10270-016-0523-3 (cit. on pp. 3, 5, 6, 15, 33, 34).

[11] B. Schätz, M. Törngreen, S. Bensalem *et al.*, "Cyber-physical european roadmap and strategy: Research agenda and recommendations for action," *CyPhERS, Tech. Rep.*, 2015 (cit. on p. 3).

[12] R. Jolak, A. Wortmann, G. Liebel, E. Umuhoza and M. R. Chaudron, "The design thinking of co-located vs. distributed software developers: Distance strikes again!" In *Proceedings of the 15th International Conference on Global Software Engineering*, 2020, pp. 106–116 (cit. on p. 3).

[13] J. Kramer, "Is abstraction the key to computing?" *Commun. ACM*, vol. 50, no. 4, pp. 36–42, 2007. DOI: 10.1145/1232743.1232745 (cit. on p. 3).

[14] J. Pinggera, P. Soffer, S. Zugal *et al.*, "Modeling styles in business process modeling," in *Enterprise, business-process and information systems modeling*, Springer, 2012, pp. 151–166 (cit. on p. 3).

[15] J. Pinggera, P. Soffer, D. Fahland *et al.*, "Styles in business process modeling: An exploration and a model," *Software & Systems Modeling*, pp. 1–26, May 2013. DOI: 10.1007/s10270-013-0349-1 (cit. on p. 3).

[16] J. Mendling, H. A. Reijers and W. M. van der Aalst, "Seven process modeling guidelines (7pmg)," *Information and Software Technology*, vol. 52, no. 2, pp. 127–136, 2010 (cit. on p. 3).

[17] J. Claes, I. Vanderfeesten, J. Pinggera, H. A. Reijers, B. Weber and G. Poels, "A visual analysis of the process of process modeling," *Information Systems and e-Business Management*, vol. 13, no. 1, pp. 147–190, 2015 (cit. on p. 3).

[18] P. Soffer, M. Kaner and Y. Wand, "Towards understanding the process of process modeling: Theoretical and empirical considerations," in *International Conference on Business Process Management*, Springer, 2011, pp. 357–369 (cit. on p. 3).

[19] J. Hutchinson, J. Whittle, M. Rouncefield and S. Kristoffersen, "Empirical assessment of mde in industry," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 471–480. DOI: 10.1145/1985793.1985858 (cit. on pp. 5, 6, 33).

[20] J. Hutchinson, M. Rouncefield and J. Whittle, "Model-driven engineering practices in industry," in *2011 33rd International Conference on Software Engineering (ICSE)*, 2011, pp. 633–642. DOI: 10.1145/1985793.1985882 (cit. on pp. 5, 6, 33).

[21] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. E. Lorensen *et al.*, *Object-oriented modeling and design*. Prentice-hall Englewood Cliffs, NJ, 1991, vol. 199 (cit. on p. 5).

[22] B. Bruegge and A. H. Dutoit, *Object-Oriented Software Engineering Using UML, Patterns, and Java*, 3rd. Prentice Hall Press, 2009, ISBN: 0136061257, 9780136061250 (cit. on p. 5).

[23] R. J. Abbott, "Program design by informal english descriptions," *Commun. ACM*, vol. 26, no. 11, pp. 882–894, 1983. DOI: 10.1145/182.358441 (cit. on p. 5).

[24] A. Maier, N. Baltsen, H. Christoffersen and H. Störrle, "Towards diagram understanding: A pilot study measuring cognitive workload through eye-tracking," in *Proceedings of International Conference on Human Behaviour in Design 2014*, 2014 (cit. on p. 5).

[25] H. Störrle, "On the impact of size to the understanding of uml diagrams," *Software & Systems Modeling*, vol. 17, no. 1, pp. 115–134, 2018. DOI: 10.1007/s10270-016-0529-x (cit. on p. 5).

[26] H. Störrle, "On the impact of layout quality to understanding uml diagrams: Size matters," in *Model-Driven Engineering Languages and Systems*, J. Dingel, W. Schulte, I. Ramos, S. Abrahão and E. Insfran, Eds., 2014, pp. 518–534 (cit. on p. 5).

[27] H. Störrle, "Diagram size vs. layout flaws: Understanding quality factors of uml diagrams," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16, 2016, 31:1–31:10 (cit. on p. 5).

[28] Q. Lohmeyer, M. Meboldt *et al.*, "How we understand engineering drawings: An eye tracking study investigating skimming and scrutinizing sequences," in *International conference on engineering design ICED*, vol. 15, 2015 (cit. on p. 5).

[29] N. Mangano, T. D. LaToza, M. Petre and A. van der Hoek, "How software designers interact with sketches at the whiteboard," *IEEE Transactions on Software Engineering*, vol. 41, no. 2, pp. 135–156, 2014 (cit. on p. 5).

[30] A. M. Madni, M. Spraragen and C. C. Madni, "Exploring and assessing complex systems' behavior through model-driven storytelling," in *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, 2014, pp. 1008–1013 (cit. on p. 5).

[31] A. M. Madni*, "Expanding stakeholder participation in upfront system engineering through storytelling in virtual worlds," *Systems Engineering*, vol. 18, no. 1, pp. 16–27, 2015 (cit. on p. 5).

[32] J. Hutchinson, J. Whittle and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Science of Computer Programming*, vol. 89, pp. 144–161, 2014, Special issue on Success Stories in Model Driven Engineering, ISSN: 0167-6423. DOI: https://doi.org/10.1016/j.scico.2013.03.017 (cit. on pp. 5, 6).

[33] J. Whittle, J. Hutchinson and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE Software*, vol. 31, no. 3, pp. 79–85, 2014. DOI: 10.1109/MS.2013.65 (cit. on pp. 5, 6, 33).

[34] M. Torchiano, F. Tomassetti, F. Ricca, A. Tiso and G. Reggio, "Relevance, benefits, and problems of software modelling and model driven techniques—a survey in the italian industry," *Journal of Systems and Software*, vol. 86, no. 8, pp. 2110–2126, 2013 (cit. on pp. 5, 6, 34).

[35] P. Mohagheghi and V. Dehlen, "Where is the proof? - a review of experiences from applying mde in industry," in *Model Driven Architecture - Foundations and Applications*, ser. Lecture Notes in Computer Science, I. Schieferdecker and A. Hartman, Eds., vol. 5095, Springer Berlin Heidelberg, 2008, pp. 432–443 (cit. on pp. 5, 33).

[36] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen and M. Fritzsche, "Where does model-driven engineering help? experiences from three industrial cases," *Software and Systems Modeling*, vol. 12, no. 3, pp. 619–639, 2013 (cit. on p. 5).

[37] P. Baker, S. Loh and F. Weil, "Model-driven engineering in a large industrial context—motorola case study," in *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2005, pp. 476–491 (cit. on pp. 5, 33).

[38] R. Kopach-Konrad, M. Lawley, M. Criswell *et al.*, "Applying systems engineering principles in improving health care delivery," *Journal of general internal medicine*, vol. 22, no. 3, pp. 431–437, 2007 (cit. on p. 6).

[39] S. Walderhaug, E. Stav and M. Mikalsen, "Experiences from model-driven development of homecare services: Uml profiles and domain models," in *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2008, pp. 199–212 (cit. on p. 6).

[40] C. Raistrick, "Applying mda and uml in the development of a healthcare system," in *International Conference on the Unified Modeling Language*, Springer, 2004, pp. 203–218 (cit. on p. 6).

[41] S. Bernonville, C. Kolski and M.-C. Beuscart-Zephir, "Contribution and limits of uml models for task modelling in a complex organizational context: Case study in the healthcare domain," in *Internet and Information Technology in Modern Organizations: Challenges & Answers, Proceedings of The 5th International Business Information Management Association Conference*, IBIMA, 2005, pp. 119–127 (cit. on p. 6).

[42] A. Schmidt, D. Kimmig, K. Bittner and M. Dickerhof, "Teaching model-driven software development: Revealing the" great miracle" of code generation to students," in *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, 2014, pp. 97–104 (cit. on p. 6).

[43] B. Westphal, "Teaching software modelling in an undergraduate introduction to software engineering," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 690–699. DOI: 10.1109/MODELS-C.2019.00105 (cit. on p. 6).

[44] F. Gilson, "Teaching software language engineering and usability through students peer reviews," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2018, pp. 98–105 (cit. on p. 6).

[45] L. Gonnord and S. Mosser, "Practicing domain-specific languages: From code to models," in *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2018, pp. 106–113 (cit. on p. 6).

[46] S. Akayama, B. Demuth, T. C. Lethbridge, M. Scholz, P. Stevens and D. R. Stikkolorum, "Tool use in software modelling education.," in *EduSymp@ MoDELS*, 2013 (cit. on pp. 7, 34).

[47] R. F. Paige, F. A. Polack, D. S. Kolovos, L. M. Rose, N. D. Matragkas and J. R. Williams, "Bad modelling teaching practices.," in *EduSymp@ MoDELS*, 2014, pp. 1–12 (cit. on p. 7).

[48] D. S. Kolovos and J. Cabot, "Towards a corpus of use-cases for model-driven engineering courses.," in *EduSymp/OSS4MDE@ MoDELS*, 2016, pp. 14–18 (cit. on pp. 7, 34).

[49] D. L. Moody, "The "physics" of notations: A scientific approach to designing visual notations in software engineering," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 2, 2010, pp. 485–486. DOI: 10.1145/1810295.1810442 (cit. on p. 7).

[50] F. Ciccozzi, G. Taentzer, A. Vallecillo *et al.*, "How do we teach modelling and model-driven engineering? a survey," in *Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems: Companion proceedings*, 2018, pp. 122–129 (cit. on p. 7).

[51] R. Reuter, T. Stark, Y. Sedelmaier, D. Landes, J. Mottok and C. Wolff, "Insights in students' problems during uml modeling," in *2020 IEEE Global Engineering Education Conference (EDUCON)*, 2020, pp. 592–600. DOI: 10.1109/EDUCON45650.2020.9125110 (cit. on p. 7).

[52] D. R. Stikkolorum, T. Ho-Quang and M. R. 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. DOI: 10.1109/SEAA.2015.77 (cit. on p. 7).

[53] L. T. Agner, T. C. Lethbridge and I. W. Soares, "Student experience with software modeling tools," *Software and Systems Modeling*, vol. 18, no. 5, 3025–3047, 2019, ISSN: 1619-1366. DOI: 10.1007/s10270-018-00709-6 (cit. on p. 7).

[54] G. Liebel, O. Badreddin and R. Heldal, "Model driven software engineering in education: A multi-case study on perception of tools and uml," in *2017 IEEE 30th Conference on Software Engineering Education and Training (CSEE T)*, 2017, pp. 124–133. DOI: 10.1109/CSEET.2017.29 (cit. on p. 7).

[55] I. Hammouda, H. Burden, R. Heldal and M. R. Chaudron, "Case tools versus pencil and paper," in *ACM/IEEE 17th Int. Conf. on Model Driven Engineering Languages and Systems–Educators Symposium*, 2014 (cit. on p. 7).

[56] M. A. Garzón, H. Aljamaan and T. C. Lethbridge, "Umple: A framework for model driven development of object-oriented systems," in *2015 ieee 22nd international conference on software analysis, evolution, and reengineering (saner)*, IEEE, 2015, pp. 494–498 (cit. on p. 7).

[57] T. C. Lethbridge, G. Mussbacher, A. Forward and O. Badreddin, "Teaching uml using umple: Applying model-oriented programming in the classroom," in *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, IEEE, 2011, pp. 421–428 (cit. on p. 7).

[58] G. Liebel, R. Heldal and J.-P. Steghöfer, "Impact of the use of industrial modelling tools on modelling education," in *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, IEEE, 2016, pp. 18–27. DOI: 10.1109/CSEET.2016.18 (cit. on pp. 7, 34).

[59] J. Whittle and J. Hutchinson, "Mismatches between industry practice and teaching of model-driven software development," in *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2011, pp. 40–47 (cit. on p. 8).

[60] J. Pinggera, P. Soffer, D. Fahland *et al.*, "Styles in business process modeling: An exploration and a model," *Software and Systems Modeling*, pp. 1–26, May 2013. DOI: 10.1007/s10270-013-0349-1 (cit. on p. 10).

[61] J. Pinggera, P. Soffer, S. Zugal *et al.*, "Modeling styles in business process modeling," in *Enterprise, Business-Process and Information Systems Modeling*, I. Bider, T. Halpin, J. Krogstie *et al.*, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 151–166, ISBN: 978-3-642-31072-0 (cit. on p. 10).

[62] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," English, Keele University and Durham University Joint Report, Tech. Rep. EBSE 2007-001, Jul. 2007 (cit. on pp. 12, 15, 16).

[63] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on software engineering*, vol. 25, no. 4, pp. 557–572, 1999 (cit. on pp. 12, 13).

[64] D. Budgen, S. Charters, M. Turner, P. Brereton, B. Kitchenham and S. Linkman, "Investigating the applicability of the evidence-based paradigm to software engineering," in *Proceedings of the 2006 international workshop on Workshop on interdisciplinary software engineering research*, 2006, pp. 7–14 (cit. on p. 12).

[65] E. Knauss, "Constructive master's thesis work in industry: Guidelines for applying design science research," in *2021 ieee/acm 43rd international conference on software engineering: Software engineering education and training (icse-seet)*, IEEE, 2021, pp. 110–121 (cit. on p. 12).

[66] M. Jøergensen, "Working with industry: Stories of successful and failed research-industry collaborations on empirical software engineering," in *2017 IEEE/ACM 5th International Workshop on Conducting Empirical Studies in Industry (CESI)*, IEEE, 2017, pp. 46–52 (cit. on p. 12).

[67] K. Petersen, R. Feldt, S. Mujtaba and M. Mattsson, "Systematic mapping studies in software engineering," in *12th international conference on evaluation and assessment in software engineering (EASE)*, BCS Learning & Development, 2008 (cit. on p. 12).

[68] K. Petersen, R. Feldt, S. Mujtaba and M. Mattsson, "Systematic mapping studies in software engineering," *Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering*, vol. 17, Jun. 2008 (cit. on pp. 12, 16).

[69] E. T. Rother, "Systematic literature review x narrative review," *Acta paulista de enfermagem*, vol. 20, pp. v–vi, 2007 (cit. on p. 13).

[70] B. Glaser and A. Strauss, *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017 (cit. on p. 13).

[71] J. Kontio, J. Bragge and L. Lehtola, "The focus group method as an empirical tool in software engineering," in *Guide to advanced empirical software engineering*, Springer, 2008, pp. 93–116 (cit. on pp. 14, 22).

[72] S. Kılıç, "Kappa testi.," *Journal of Mood Disorders*, vol. 5, no. 3, 2015 (cit. on p. 14).

[73] K.-J. Stol and B. Fitzgerald, "The abc of software engineering research," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 27, no. 3, pp. 1–51, 2018 (cit. on p. 18).

[74] J. Saldaña, *The coding manual for qualitative researchers*. SAGE Publications, 2015, ISBN: 9781473902497 (cit. on p. 19).

[75] L. Zhang, J.-H. Tian, J. Jiang, Y.-J. Liu, M.-Y. Pu and T. Yue, "Empirical research in software engineering — a literature survey," in *Journal of Computer Science and Technology*, 2018, pp. 266–276. DOI: 10.1007/s11390-018-1864-x (cit. on p. 25).

[76] K. Oliveira, A. Regina, C. Rocha, G. Travassos and C. Menezes, "Using domain-knowledge in software development environments," CiteSeerX, Tech. Rep., May 2000. [Online]. Available: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.2007 (cit. on p. 27).

[77] S. Rugaber, "The use of domain knowledge in program understanding," *Annals of Software Engineering*, vol. 9, no. 1, pp. 143–192, 2000 (cit. on p. 28).

[78] S. Chakraborty and G. Liebel, "We do not understand what it says–studying student perceptions of software modelling," *Empirical Software Engineering*, vol. 28, no. 6, p. 149, 2023 (cit. on p. 29).

[79] J. S. Krajcik and P. C. Blumenfeld, *Project-based learning*. na, 2006 (cit. on p. 30).

[80] M. Seidl, M. Scholz, C. Huemer and G. Kappel, *UML@ classroom: An introduction to object-oriented modeling*. Springer, 2014 (cit. on p. 31).

[81] L. Nóbrega, N. J. Nunes and H. Coelho, "The meta sketch editor: A reflexive modeling editor," in *Computer-Aided Design Of User Interfaces V*, Springer, 2007, pp. 201–214 (cit. on p. 33).

[82] P. Pourali and J. M. Atlee, "An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools," in *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS '18, Copenhagen, Denmark: Association for Computing Machinery, 2018, 224–234, ISBN: 9781450349499. DOI: 10.1145/3239372.3239400 (cit. on p. 33).

[83] S. Bimonte, K. Boulil, F. Pinet and M.-A. Kang, "Design of complex spatio-multidimensional models with the icsolap uml profile-an implementation in magicdraw," in *International Conference on Enterprise Information Systems*, SCITEPRESS, vol. 2, 2013, pp. 310–315 (cit. on p. 33).

[84] T. Pati, S. Kolli and J. H. Hill, "Proactive modeling: A new model intelligence technique," *Software & Systems Modeling*, vol. 16, pp. 499–521, 2017 (cit. on p. 33).

[85] J. E. Robbins and D. F. Redmiles, "Cognitive support, uml adherence, and xmi interchange in argo/uml," *Information and Software Technology*, vol. 42, no. 2, pp. 79–89, 2000 (cit. on p. 33).

[86] 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 (cit. on pp. 33, 34).

[87] H. Störrle, "How are conceptual models used in industrial software development? a descriptive survey," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 160–169 (cit. on p. 33).

[88] J. Whittle, J. Hutchinson, M. Rouncefield, H. Burden and R. Heldal, "Industrial adoption of model-driven engineering: Are the tools really the problem?" In *International Conference on Model Driven Engineering Languages and Systems*, Springer, 2013, pp. 1–17 (cit. on pp. 33, 34).

[89] J. Whittle and J. Hutchinson, "Mismatches between industry practice and teaching of model-driven software development," in *Models in Software Engineering: Workshops and Symposia at MODELS 2011, Wellington, New Zealand, October 16-21, 2011, Reports and Revised Selected Papers 14*, Springer, 2012, pp. 40–47 (cit. on p. 34).

[90] A. Akundi and W. Ankobiah, "Mapping industry workforce needs to academic curricula–a workforce development effort in model-based systems engineering," *Systems Engineering*, 2024 (cit. on p. 34).

[91] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012 (cit. on pp. 37–39).

[92] J. Saldana, *The coding manual for qualitative researchers*, 3rd ed. SAGE Publications, 2015, ISBN: 9781473902497 (cit. on p. 37).

[93] P. Runeson, M. Höst, A. Rainer and B. Regnell, *Case Study Research in Software Engineering – Guidelines and Examples*. John Wiley & Sons Inc., Feb. 2012. DOI: 10.1002/9781118181034 (cit. on p. 38).

[94] J. Hiller, *Epistemological foundations of objectivist and interpretivist research*. Barcelona Publishers, 2016 (cit. on p. 38).

[95] C.-M. Pascale, *Cartographies of knowledge: Exploring qualitative epistemologies*. Sage Publications, 2010 (cit. on p. 38).

[96] R. K. Yin, *Case study research: Design and methods*. sage, 2009, vol. 5 (cit. on p. 38).

[97] K. Petersen and C. Gencel, "Worldviews, research methods, and their relationship to validity in empirical software engineering research," in *2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement*, IEEE, 2013, pp. 81–89 (cit. on p. 38).

[98] D. Strüber, "The complexity paradox: An analysis of modeling education through the lens of complexity science," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, IEEE, 2023, pp. 94–97 (cit. on p. 41).

Part II

Appended Papers

Chapter 6

Paper I: Modelling guidance in software engineering: a systematic literature review

Shalini Chakraborty, Grischa Liebel
Software and Systems Modeling (2023), 1-17
https://doi.org/10.1007/s10270-023-01117-1

Software and Systems Modeling (2024) 23:249–265
https://doi.org/10.1007/s10270-023-01117-1

REGULAR PAPER

Modelling guidance in software engineering: a systematic literature review

Shalini Chakraborty[1] · **Grischa Liebel**[1]

Received: 15 June 2022 / Revised: 12 June 2023 / Accepted: 27 June 2023 / Published online: 17 July 2023

Abstract
Despite potential benefits in Software Engineering, adoption of software modelling in industry is low. Technical issues such as tool support have gained significant research before, but individual guidance and training have received little attention. As a first step towards providing the necessary guidance in modelling, we conduct a systematic literature review to explore the current state of the art. We searched academic literature for guidance on model creation and selected 35 papers for full-text screening through three rounds of selection. We find research on model creation guidance to be fragmented, with inconsistent usage of terminology, and a lack of empirical validation or supporting evidence. We outline the different dimensions commonly used to provide guidance on software and system model creation. Additionally, we provide definitions of the three terms modelling method, style, and guideline as current literature lacks a well-defined distinction between them. These definitions can help distinguishing between important concepts and provide precise modelling guidance.

Keywords Modelling styles · Modelling training · Modelling guidance · Modelling method · Systematic literature review

1 Introduction

Despite the potential benefits of using models in Software Engineering (SE), adoption has been low, typically pointing to issues such as tool support, organisational resistance, and a lack of guidance/training [39–41, 53, 54, 67, 68, 97]. Technical issues have historically received substantial attention in the modelling community, seen for instance by the large amount of tools for modelling and Model-Based Engineering (MBE). However, work on guiding individuals in creating models receives typically only marginal attention. For instance, Schätz et al. state that "methodical guidelines are missing how to use suitable abstractions of (parts of) a cyber-physical systems at varying level of detail to enable the engineering of those systems with a sufficient level of confidence concerning the quality of the implemented systems" [84].

Communicated by Juergen Dingel.

Shalini Chakraborty
shalini19@ru.is

Grischa Liebel
grischal@ru.is

[1] Reykjavik University, Menntavegur 1, 102 Reykjavík, Iceland

To better understand shortcomings in the literature on guidance and training in modelling, this paper reports on a Systematic Literature Review (SLR) [48] surveying work on guidelines, styles, or training approaches on creating software and system models in SE. We address the following research question **RQ: What kind of guidance exists in SE literature on model creation?** To answer the question, we collected a total of 8604 papers starting from 1998 when the first UML conference was held. In addition to this database search, we conducted a second search using more general terminology and random sampling among the 71450 resulting papers, and snowballing [99] in the area of Business Process Modelling to complement our findings.

After several rounds of exclusion, we analysed 35 papers, finding that systematic guidance to create software and system models is limited in SE literature. Existing work proposes guidance for specific domains or problems, but generally lacks validation and/or empirical evidence. In BPM, which partially overlaps with SE research, there exists initial empirical work investigating modelling styles and how cognitive processes affect model creation. We further find that terminology is used inconsistently in the literature, with terms such as method, guidelines, or style being used inconsistently and seemingly arbitrary. To address this, we define *modelling method*, *modelling guideline*, and *modelling style*.

Our SLR shows that further work is necessary in several directions in the modelling community. First, we see that work on modelling needs to be conducted more systematically. Our definitions presented in this paper can help supporting reaching a common terminology. Secondly, empirical evidence and validation is needed. Currently, we find mainly solution proposals that are at best substantiated through simplified examples. The data set accompanying this paper is published on Zenodo.[1]

The rest of the paper is structured as follows. In Sect. 2, we discuss the lack of guidance in software modelling through related work. Section 3 explains the methodology of our conducted SLR and validity threats. We present the results in Sect. 4, followed by an overall discussion of the findings in Sect. 5. Finally, we conclude the paper in Sect. 6 with future plans to use the findings suitably in software modelling.

2 Related work

Text books on UML and object-oriented design commonly refer to heuristics for creating UML diagrams, e.g. [9, 79]. For instance, Abbott's [1] strategy for identifying system objects by scanning informal descriptions for nouns is typically referred to. These and similar heuristics are common for some diagram types, e.g. class diagrams, but often lacking for behaviour. Additionally, there is, to our knowledge, no empirical evidence that these kind of heuristics support system analysis or other modelling tasks.

Previous work has investigated how to design modelling languages in a *good* way, by studying both the notations themselves, e.g. [69, 82, 83, 92], and how models or diagrams are read and understood by human subjects, e.g. [55, 60, 89–91]. Moody [69] defines in his Physics of Notation a number of principles for cognitively effective notations. However, the value of the Physics of Notation is unclear [82, 83, 92]. For diagram understanding, Störrle et al. [60, 89–91] find that the size and layout quality of UML diagrams directly impacts understanding. Outside of SE, Lohmeyer and Meboldt [55] find that there are different patterns individuals follow when reading engineering drawings. Understanding how models are read and understood can help to derive heuristics for the quality of a model. However, this knowledge might not help guiding individuals to create *good* models. That is, while the desired outcome might be known, the process to arrive at this outcome remains unclear.

A way to describe how to create models is to provide reusable parts of a solution, or patterns, for different kinds of models. In the area of patterns, there has been substantial research during the late 1990 s, e.g. [7, 20–22, 29]. The well-known Gang-of-Four design patterns [29] provide reusable patterns that should help to improve the design of object-oriented systems. Douglass describes patterns for real-time behaviour diagrams [20], and Bordeleau [7] proposes patterns that should aid a designer to move from a scenario-based specification to a state-based specification. Dwyer et al. [21, 22] propose property specification patterns, reusable logic constructs that can be re-used to specify properties in specifications, specifically the order and the occurrence of system events. The work on patterns has evolved substantially since then, especially in the area of real-time systems, e.g. [3, 33, 61]. However, patterns describe only a part of model creation, namely *what* should be part of the model. There are several other aspects that could be of interest during model creation, e.g. in which order to create parts of the model, what to include and what to exclude, how to decide on a reasonable level of abstraction, and individual differences that affect this process. A promising area of interest is AI-assisted modelling guidance. Recent studies have shown the application of AI in model creation, leading to enhanced functional objectives. Cámara et al. [12] describe the role of generative AI models such as large language models (LLM) in software modelling regulation. Chaaben et al. [13] propose using LLM to improve domain modelling activities. A recent special issue on AI-enhanced model-driven engineering [10] also summarises several contributions [81, 93, 96] to support modellers with better search facilities, automated integration and better learning experience from independent metamodels. While clearly relevant for future modelling support, we excluded AI-assisted work in our literature review, as these approaches as of now do not provide explicit and explainable guidance, but rather simply improve the resulting models.

In terms of guiding individual modellers and understanding the differences between individuals when creating models, Pinggera et al. [15, 74, 75] investigate BPM model creation. The authors present an exploratory study with 115 students exploring how students create BPM. They find distinct styles of modelling by performing cluster analysis on phases of model comprehension (where a modeller builds a mental model out of the domain behaviour), modelling (where a modeller maps the mental model to actual modelling constructs), and finally reconciliation (where a modeller acknowledges the process model).

3 Research methodology

In this paper, we aim to answer the following research question:

RQ: What kind of guidance exists in SE literature on model creation?

To do so, we conduct an SLR, a secondary form of study used to identify and evaluate available research relevant to a certain

[1] https://doi.org/10.5281/zenodo.7685694.

research question or topic of interest [48]. SLRs are useful to obtain a detailed picture of a research area, and to integrate existing evidence [49]. They are increasingly common in SE [47], and specifically in the modelling community in the last decade, e.g. [30, 57, 70, 71, 87]. We chose an SLR over a mapping study, as we were interested in the detailed picture, not the broad coverage of a research area a mapping study provides [72].

We followed the steps proposed by Kitchenham and Charters [48] to perform our SLR. An overview of this process is depicted in Fig. 1.

In the following, we discuss the review steps in detail, starting with the search and extraction process.

3.1 Search and extraction

We started the review with preliminary research steps, by manually reading through the last five years (2015 to 2019 at the time of starting the review) of papers appearing in *MODELS conference* (http://modelsconference.org/) and *SoSyM journal* (https://link.springer.com/journal/volumesAndIssues/10270), the two prime venues for SE modelling research. Note that the automated keyword search was later updated until January 2023. We observed that papers use the terms *framework*, *guidelines*, *method*, *approach*, and *pattern*. Hence, we used these terms in our initial search string. We searched within modelling research in SE, thus resulting in the following initial search string:

(“modelling” OR “modelling” OR “model-driven” OR “model-based”)
AND (“framework” OR “guidelines” OR “method” OR “approach” OR “pattern”)
AND (“software engineering”)

Applying the initial search string on Scopus, IEEE Xplore, ACM Digital Library, and Web of Science resulted in 17713 papers, a result we deemed infeasible to analyse. Therefore, we decided to make a number of adaptations to the search string (Step 1 in Fig. 1). First, we added three terms we had missed initially: *creation*, *training*, and *styles*. Adding creation and training is directly motivated by our research question and goal, while adding styles is motivated by existing work in BPM, i.e. by Pingerra et al. [74]. Secondly, to reduce the amount of found papers, we decided to remove *method*, *approach*, and *pattern*. This resulted in the following final search string **S1**:

(“modelling” OR “modelling” OR “model-driven” OR “model-based”)
AND (“guidelines” OR “training” OR “styles” OR “creation”)
AND (“software engineering”).

To not entirely discard relevant search terms, we constructed a second search term **S2** as follows:

(“modelling” OR “modelling” OR “model-driven” OR “model-based”)
AND (“approach” OR “process” OR “method” OR “template”)
AND (“software engineering”).

S2 contains terms that are heavily conflated in SE research. That is, we expect less relevant results from **S2**. Therefore, combining **S1** and **S2** would have required to do random sampling on the whole data set, and resulted in a higher chance to miss relevant papers. We decided to extract all papers found through **S1**, while randomly sampling from **S2** up to a feasible amount of papers.

We then adapted the two search strings to the four search engines we used, further limiting the search to English papers published from 1998, as the first UML conference was held in that year [6]. We followed the UK spelling of modelling throughout our review, but we used both UK and US spelling in our search string. Additionally, we added *model-driven* and *model-based* to target papers specifically aimed towards model-driven and model-based engineering (MDE & MBE) as well.

We searched in title, abstract, and keywords for Scopus, IEEE Xplore, and ACM Digital Library, and in all fields offered by Web of Science (Step 2 in Fig. 1). For ACM Digital Library, we had to further exclude the keyword *creation*, since one of the standardised ACM keywords (“CCS concepts”) is coined “Software creation and its management.” As it was not possible to exclude the CCS concepts from the search, we would get all papers that used this keyword prior to removing *creation* from the search. The search for **S1** yielded 9905 papers, 8604 after removing duplicates, covering the years from 1998 until January 2023. The search for **S2** yielded a staggering 78243 papers, 71450 after duplicate removal. We decided that sampling 5 per cent of these papers would be reasonable, as it both is a substantial amount of papers, and one we could realistically process without experiencing fatigue bias. That is, we randomly selected 3572 papers from the results of **S2**.

3.2 Paper selection

We performed paper selection in three rounds with different inclusion and exclusion criteria for each round (Step 3a in Fig. 1). In the beginning of each round, both authors went through a random 5 per cent of the remaining papers in **S1** to determine inter-rater agreement on the respective inclusion/exclusion criteria. We used Fleiss kappa to measure our agreement, a statistical measure used to evaluate the reliability of an agreement between a fixed number of raters [46]. We decided on a threshold value of $\kappa > 0.7$ as a minimum agreement to continue the selection process. This is a compromise between Fleiss [27] suggestion to understand kappa values of > 0.75 as excellent, and suggestions of Landis and Koch

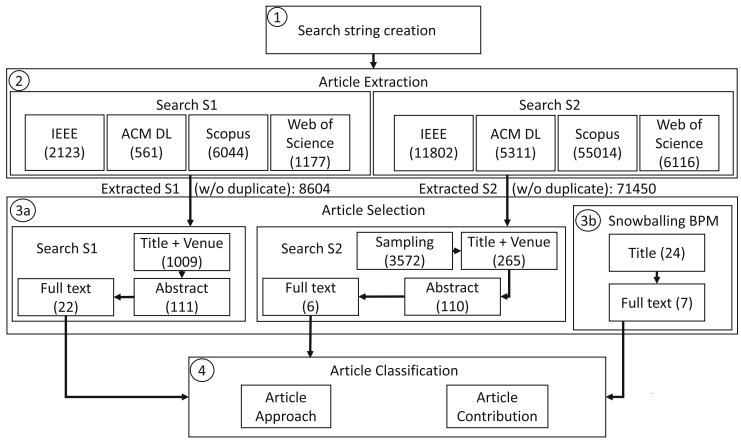


Fig. 1 Review Process

[50] to label agreements between $0.61 - 0.80$ as substantial, both of which have been criticised for being arbitrary [36]. In case of lower values, we would discuss our disagreements, potentially refine the criteria, and then re-run the process with another random selection of papers.

In the first round, we excluded papers based on title and the paper venue. We needed three rounds to reach a Fleiss Kappa of $\kappa \approx 0.73$. Ultimately, we excluded papers in the first round if they matched any of the following criteria. Note that we applied the criteria extremely conservative at this stage, to not exclude relevant papers.

1. It is clear from the title that modelling/diagrams are not a contribution.
2. The conference/journal is from a different field of science.
3. The paper is not peer reviewed.
4. The paper discusses properties of modelling languages or compares languages.
5. The paper has a prose title, or clearly points into a different direction than investigated in our review.
6. The paper is about descriptive models of a scientific or real-life concept, as opposed to a system. For instance, process improvement models such as CMMI, or models of course curricula.
7. Modelling in a different domain of computer science, e.g. database models, machine learning models, or neural network models.
8. Extended abstracts for tool demos or posters.
9. Meta studies such as literature reviews, or comparative studies such as controlled experiments.

After applying these criteria, we were left with 1009 papers of S1 and 265 papers of S2 for the second round of selection.

In the second round, we considered the paper abstracts. After two rounds of discussions and adapting the exclusion criteria, we reached a kappa of $\kappa \approx 0.79$.

The exclusion criteria for excluding based on abstracts are as follows.

1. Creation of models is not clearly mentioned in the abstract.
2. It is clear from the abstract that the paper focuses on any of the following.
   (a) Language design/meta modelling
   (b) Model transformations
   (c) Secondary studies such as SLRs
   (d) Controlled experiments or other comparative studies
   (e) Descriptive models of a scientific or real-life concept, as opposed to a software system
   (f) Architectural styles or design patterns
   (g) AI-based modelling guidance

We excluded language design/meta modelling and model transformations, as we consider them special cases of modelling aimed at the meta modelling level. That is, they aim at creating models that describe languages and/or domains, and not software and/or systems. As such, it can be expected that the resulting guidance would be substantially different from models that describe systems. For similar reasons, we excluded architectural styles and design patterns, as they present tried solutions, but do not provide guidance in any other way. Finally, we excluded papers that use AI to improve or create models, as they lack explicitly stated guidance [23, 51]. As such, while valuable, their output is often not explainable. For example, we found papers like [45, 77] where modelling frameworks are mentioned, but no particular steps for modelling are introduced. The papers we found through

our search that include AI-related guidance mostly detailed with the model's impact on systems rather than creation. Applying these criteria, 111 papers from S1 and 110 papers from S2 remained for full-text reading.

After the second exclusion round, we found three papers from the area of BPM that fit into the scope of our review. However, they were not included through the search, as they did not fit the software engineering keyword. Nevertheless, to be able to compare SE-specific research with promising work in BPM (but outside SE), we took those three papers as a starting set for backwards and forwards snowballing [99] (Step 3b in Fig. 1). That is, we searched both the references and citations of the three papers, including papers that fit our scope. We used the same exclusion criteria as for the second round resulting in a set of 24 papers from BPM.

In the third and last round, we extracted and read the full-text of the 111 papers from S1 and 110 papers from S2, as well as the 24 papers from the snowball search in BPM. We used the same exclusion criteria as for the second round. We applied the criteria to each paper's full text. Furthermore, we excluded one paper, as an extended version of that paper was already included in the review.

This resulted in a final set of 28 papers from S1 and S2, and 7 papers from the snowball search left for analysis.

3.3 Data extraction

The final 35 papers we analysed are listed in Table 7 for S1, Table 9 for S2, and Table 8 for the snowballing search in Appendix A. We provide the citations of all 35 papers in Appendix A. Both authors analysed all 35 papers, then discussed their disagreements. We did not calculate inter-rater agreement at this step. We deemed this unnecessary, as re-running the analysis several times would likely result in the authors memorising the discussions, not an actual assessment of inter-rater agreement.

For analysis, we classified each paper based on their **research approaches** and **research contributions** (Step 4 in Fig. 1). We use the classification of research approaches by Wieringa et al. [98] and later updated by Petersen et al. [73] to describe the contribution of the papers and thus the maturity of the research area. Table 1 summarises the different paper categories.

To provide a detailed view on what parts of modelling the papers contribute to, we break down the papers into different *modelling concerns*. These concerns are based on a classification we developed in previous work [52]. We do not claim that these concerns are exhaustive, but they serve as a useful tool to further break down a paper's contribution in terms of modelling. The modelling concerns are depicted in a basic feature model notation in Fig. 2. Solid lines from the *modelling concerns* box with an empty circle at the end depict optional features. That is, all of the features on the right-hand side of the figure are potential concerns of modelling that can be discussed in a paper. However, none is mandatory, as all features can be omitted in a paper. The different concerns are as follows:

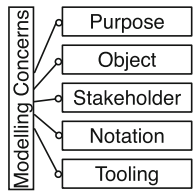


Fig. 2 Classification of Modelling Concerns. Adapted from [52]

1. Purpose: For what purpose is/are the model(s) created? For instance, models could serve as an input for code generation, or as an architecture description.
2. Object: What is being described in the model? For instance, the model could describe an entire system, a subsystem or component, or a process.
3. Stakeholder: Who is/are the primary stakeholders of the model? For example, a model could primarily serve as end-user documentation, or as a blueprint for developers.
4. Notation: What modelling notation(s) is/are used? For instance, UML might be used.
5. Tooling: Which tools are prescribed? For instance, Eclipse Papyrus might be prescribed due to a dependence on a custom plugin.

To understand the modelling concerns, we used open coding in the 35 papers.

3.4 Threats to validity

In this section, we describe the potential validity threats of our study and the steps we have taken to mitigate them. We follow the categorisation of threats into internal validity, construct validity, external validity, and reliability according to Runeson et al. [80].

3.4.1 Internal validity

Internal validity reflects to what extent causal relationships are closely examined and other, unknown factors might impact the findings.

As a main means to reduce threats to internal validity, we tried to increase reliability of our exclusion steps by calculating inter-rater agreements and repeating the steps until a satisfactory level of agreement was reached between the two authors. To avoid memory effects, we chose a new random sample to calculate inter-rater agreements after each round.

A threat of our review protocol is the strictness of the exclusion criteria. As we had to reduce a large number of initial papers to a manageable amount for detailed analysis,

Table 1 Classification of paper types based on Wieringa et al. [98] and Petersen et al. [73]

Research approach	Definition
Evaluation research	Investigation of a problem employing case study, controlled experiment, survey etc in the industrial context, implementing a technique in practice, and evaluating the implementation, i.e. showing the benefits and drawbacks of the implementation
Solution proposals	Proposed solution (either novel or extension of an existing technique) with a relevant argument or a good example to show the solution's benefits. No empirical evaluation
Validation research	Novel research techniques used for experiments which are not established in practice (for example, studies conducted with students); investigation of properties of a solution proposal that has not been implemented yet
Philosophical papers	A conceptual framework that sketches a new way of looking at existing things
Opinion papers	An author's personal opinion whether a certain technique is good or bad, or how things should be done
Experience papers	An author's personal experience of how something has been done

and furthermore had to deal with ambiguous terminology, we decided to exclude strongly in the first two exclusion rounds (based on title and venue, and based on abstract). This could lead to papers being excluded that do contain valuable guidance to model creation, but do not clearly state so in their abstracts. Given the trade-off between search coverage and detailed analysis, we decided to accept this threat. In particular, we observe that venues such as Requirements Engineering (RE) conference and the European Conference on Modelling Foundations and Applications (ECMFA) are not represented in the data set, despite strong focus on modelling. This might be due to a different focus, e.g. on theoretical aspects in ECMFA, but we do not know whether this could represent a bias.

3.4.2 Construct validity

Construct validity reflects to what extent the measures represent the construct investigated.

In this paper, we investigate model creation, guidance, and related topics. A threat to the validity of our study is that these topics are not described using established, standardised terminology in the modelling community. Therefore, it might in some cases be difficult to decide when a paper in fact provides guidance for model creation, and when not. To reduce this threat, we used open coding for data analysis, without relying on fixed keywords being used in the respective papers.

3.4.3 External validity

External validity is reflecting to what extent findings can be generalised beyond the concrete sample.

In case of our literature review, we use the major digital libraries for data collection, which should lead to a representative sample of publications in software engineering. A potential threat to validity in the review was our decision to treat the search string S2 separate from S1, due to the conflated keywords "approach," "method," "process" and "template." The relative amount of included papers is somewhat similar in S1 and S2, indicating that a lot of relevant papers might have been excluded by performing random sampling on S2. This is a threat we have to accept given the large amount of search results. However, we also note that the papers from S2 did not yield any results substantially different from the papers in S1.

3.4.4 Reliability

Reliability describes the degree to which similar results would be obtained if the same study would be repeated, by the same or by other researchers.

The different steps of the data collection and analysis are clearly described in the previous subsections, so that we are confident that other researchers could repeat the study. Nevertheless, there is subjectivity to several parts of our study, in particular the exclusion and analysis steps of the review. We tried to be conservative in our exclusion, and calculated inter-rater agreement at all exclusion steps. Furthermore, the analysis was performed by multiple researchers, and disagreements were discussed. To increase reliability, the data set with detailed tables is published[2]).

4 Results

In this section, we present the results of the SLR. First, we classify the final 35 papers in Sect. 4.1. Based on the little empirical evidence, we find in the modelling literature, we then introduce definitions for modelling guidelines, methods and styles in Sect. 4.2. Finally, we answer the RQ in Sect. 4.3.

[2] https://doi.org/10.5281/zenodo.7685694.

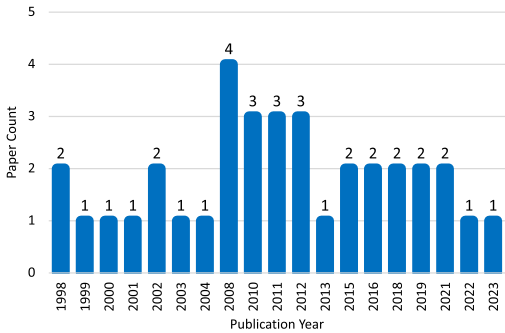


Fig. 3 Paper count per year. Years without papers are omitted

Table 2 Paper types in our analysis (based on Wieringa et al. [98])

Paper type	# papers
Evaluation research (EV)	8
Solution proposals (SP)	22
Validation research (VR)	1
Philosophical papers (PH)	1
Opinion papers (OP)	0
Experience papers (EX)	3

4.1 Paper classification

We ultimately analysed 22 papers through **S1**, with an additional 7 papers added through snowball search and 6 papers added through **S2**. The publication years of our final 35 papers range from 1998 to 2023, with the majority of the papers from 2008 to 2015. Figure 3 depicts the actual counts, where years without paper are omitted.

There is no clear pattern with respect to publication venues in the data. Venues targeted towards modelling, such as the ER and UML/Models conference series or the Software and Systems Modelling journal are present, as well as general software engineering venues, such as ICSE or Transactions on Software Engineering.

Table 2 summarises the types of the 35 papers.

In terms of modelling concerns, we find that the model purpose, the modelling notation, and the object that is modelled are typically described in the papers, or can be derived from the descriptions. In contrast, stakeholders and tooling are rarely reported.

The included papers state several *purposes* for the models created. Most commonly (15 out of 35 papers), there is no exact purpose stated for the constructed models, i.e. the purpose is to model parts of or a complete system. The guidance provided in the papers thus aims broadly at improving the quality of the resulting models, or make the process more systematic. The remaining papers state specific purposes for their constructed models. These purposes vary widely, e.g. models for simulation, formal verification, evaluation of any system property (e.g. security), requirements and system analysis, avoiding inconsistencies between models, improving communication between teams and generate (parts of) an application. Given our snowball search in the area of BPM, business processes are the dominating *object* modelled in the included papers. This is followed by six papers where modelling of embedded control systems is reported. One paper describes modelling to address many challenges during the design of Space Interferometry Mission (SIM). The remainder of the papers focuses on a mix of different models, such as use cases, architecture, and requirements.

Only two papers report a specific *stakeholder*. Additionally, several papers refer to general terms such as *engineers, programmers, users, domain experts, or software developers*. That is, it is often unclear who creates, modifies, reads, or otherwise comes in contact with the proposed models.

15 of our 35 papers discuss models in UML *notation*, or extensions thereof, such as UML profiles or the UML-RT extensions. This is followed by 6 papers using BPMN. Finally, several papers do not focus on a specific notation, or propose their own notations.

Tools are rarely reported in our data set. Only 7 out of 35 papers report a specific tool, 6 of which are specialised tools developed for the presented approaches and one paper uses FAME toolchain. Of the remaining 28 papers, most are tool agnostic, e.g. as they target general quality improvement to a specific type of model or as they do not rely on specific tool features.

4.2 Definitions of key terms in model creation

Starting from the initial keyword search and continuing throughout the entire analysis, we noticed that terms such as *method*, *guidelines*, and *style* are used seemingly arbitrary in the community. Therefore, we decided to provide definitions for a number of key terms, in order to allow us to better structure the literature, and to support future work in the area.

Modelling is a creative task, which requires thinking and problem solving, both of which can affect the quality of the model. In the general literature on problem solving, Isaksen et al. [42] discuss that, "...when individuals, in both school and corporate settings, understand their own style of problem solving, they are able to learn and apply process tools more effectively, and when teams appreciate the styles of their individual members, their problem solving efforts are enhanced". That is, styles in problem solving differ between individuals, which makes it necessary to distinguish from guidelines or methods, where specific steps or activities are prescribed on good practice.

We define the term *modelling style* in reference to cognitive styles, which are "an individual's preferred way of

gathering, processing and evaluating data" [2]. Messik [64] defines cognitive styles as "consistent individual differences in preferred ways of organising and processing information and experience." Accordingly, we define modelling styles as follows.

> **Modelling styles** are "consistent individual differences in preferred ways of creating and processing software models."

Here, creation can include aspects such as the choice of modelling language, preferred elements of those languages, of tools, and organising the created model (i.e. layouting, or using a specific subset of modelling elements from a given language). Processing includes reading, understanding and validating models. For example, styles could be "I like to draw all classes first, then connect them" or "I like to have the most general classes in the top."

In contrast, we define modelling methods as follows.

> **Modelling methods** are "sets of steps or constraints on how one or multiple models are created."

In contrast to a modelling style, a modelling method is identical across individuals, and does not depend on preferences. However, modelling methods and modelling styles overlap and may conflict with each other, e.g. if an individual's preferred ways of creating a model are in conflict with the prescribed method. For instance, the method proposed by Machado et al. [59] proposes a step-wise formalisation of behaviour, starting from textual rules. This could be in conflict with an engineer who works by incrementally adding formal behaviour models, entirely omitting (informal) textual input. Another example of a modelling method would be the steps of creating the diagram such as "first underline all nouns, then underline verbs that relate two or more nouns. Then use the resulting nouns as classes, and the verbs as relations." (adapted from the classic recommendations by Abbott [1]).

Finally, for guidelines we use an existing dictionary definition.

> **Guidelines** are "information intended to advise people on how something should be done or what something should be[3]".

Following this definition, modelling methods are always also guidelines, as they advise people how to create a model. The opposite is not true, as guidelines do not necessarily have to be in the form of steps or constraints. An example set of guidelines is provided by Reggio et al. [76], who recommend which parts of UML activity diagrams should be used depending on the purpose of the resulting model. A simplified example for guidelines could be instructions such as "do not include the user in a class diagram" or "only use classes that do not refer to implementation details."

Note that the definitions for methods and guidelines do not include the purpose they serve in detail. For instance, guidelines could be used to provide recommendations on how to use a notation given a chosen task, such as in [76]. Similarly, providing recommendations on naming, as in [103], could help creating models that are easily communicated and maintained in an organisation, i.e. a form of standardisation. While we do not include it in our definitions, we believe providing clear purpose statements increases the usefulness of modelling guidance.

4.3 Modelling creation guidance in literature

After defining the terms modelling *style*, *method*, and *guidelines* in Sect. 4.2, we now re-visit the literature in terms of our RQ, i.e. *What kind of guidance exists in SE literature on model creation?*.The included papers and their corresponding information are described in Tables 3, 4, 5, and 6.

Based on the definitions, we found 11 papers containing guidelines (i.e. [5, 16–18, 25, 26, 35, 63, 76, 85, 102]), 20 papers describing modelling methods (i.e. [11, 19, 24, 31, 32, 34, 37, 38, 43, 44, 58, 59, 62, 65, 66, 78, 88, 95, 100, 103]), and 1 paper investigating modelling styles (i.e. [75]). Out of the remaining 3 papers, 1 (i.e.[28]) includes both method and a guideline. The rest (i.e. [14, 86]) discuss selected modelling concerns without providing any of the above. In the following, we describe different concerns addressed by the included papers, and relate these to whether guidelines, methods, or styles are described.

4.3.1 Inconsistent terminology

As discussed in Sect. 4.2, terminology is used inconsistently across the literature. For instance, [75] describes modelling style in a similar way as we defined it above, while the "styles" described in [76] are different levels of formality used in business process models. That is, they do not refer to individual differences between modellers. Similarly, the described "method" in [76] is not a set of steps or constraints and therefore rather falls under our definition of guidelines. We also find "guidelines" described in [31] that fit our description of a method. Finally, we see a mixed approach consisting of a step-wise "prescribed process" with a list of "guidelines" to apply those steps in [28].

[3] https://dictionary.cambridge.org/dictionary/english/guideline?q=Guidelines.

Table 3 Paper overview, Part 1

Paper	[43]	[59]	[38]	[24]	[19]	[25]	[76]	[35]	[31]
Guidance type	M	M	M	M	M	G	G	G	M
Paper type	SP	SP	SP	SP	SP	EX	SP	SP	SP
What versus how	Both	What	Both	What	What	What	What	What	Both
Views/perspectives	No	No	Yes	Yes	Yes	No	No	No	No
Decomposition/refinement	Yes	Yes	No	No	Yes	Yes	No	Yes	No
Practices/anti-patterns	No	No	Yes	No	No	No	Yes	No	No
Cognition	No	No	No	No	No	No	No	No	No

Guidance type consists of Methods (M), Guidelines (G), and Styles (S). Paper types are listed according to Table 2

Table 4 Paper overview, Part 2

Paper	[85]	[62]	[18]	[102]	[5]	[95]	[100]	[26]	[44]
Guidance type	G	M	G	G	G	M	M	G	M
Paper type	SP	SP	EV	EV	SP	SP	EV	EX	SP
What versus how	Both	What	How	How	What	What	Both	Both	What
Views/perspectives	No	No	No	No	No	Yes	No	No	No
Decomposition/refinement	No	Yes	No	No	No	Yes	No	No	No
Practices/anti-patterns	Yes	No	Yes	Yes	Yes	No	No	Yes	No
Cognition	No	No	No	No	No	No	No	No	No

Guidance type consists of Methods (M), Guidelines (G), and Styles (S). Paper types are listed according to Table 2

Table 5 Paper overview, Part 3

Paper	[65]	[16]	[75]	[14]	[86]		[63]	[17]	[32]
Guidance type	M	G	S	N/A	N/A	M	G	G	M
Paper type	EV	SP	EV	PH	VR	EV	EV	SP	SP
What versus how	How	How	N/A	N/A	N/A	How	How	Both	What
Views/perspectives	No	Yes	N/A	N/A	N/A	No	No	No	No
Decomposition/refinement	Yes	Yes	N/A	N/A	N/A	Yes	No	No	No
Practices/anti-patterns	No	No	N/A	N/A	N/A	No	Yes	Yes	No
Cognition	No	No	Yes	Yes	Yes	No	No	No	No

Guidance type consists of Methods (M), Guidelines (G), and Styles (S). Paper types are listed according to Table 2

Table 6 Paper overview, Part 4

Paper	[37]	[28]	[103]	[58]	[88]	[11]	[34]	[66]
Guidance type	M	M	M	M	M	M	M	M
Paper type	SP	SP	SP	SP	EV	SP	SP	EX
What versus how	Both	How	Both	What	What	Both	What	Both
Views/perspectives	No	No	No	No	Yes	No	No	No
Decomposition/refinement	Yes	Yes	Yes	No	Yes	Yes	No	Yes
Practices/anti-patterns	No	Yes	Yes	No	No	No	No	Yes
Cognition	No	No	No	No	No	No	No	No

Guidance type consists of Methods (M), Guidelines (G), and Styles (S). Paper types are listed according to Table 2

4.3.2 What instead of how

There is one major distinction that can be made between our included papers. While some papers give concrete advice on how a single model (or models of the same type) should be created, 25 papers provide guidance on using different models/diagrams towards an aim, such as modelling an entire system or specific property of a system. That is, the latter type of papers typically focuses more on *what* to use, e.g. in terms of diagram types. As timing and order are important elements in these descriptions, several of these papers are method papers (18 out of 25).

For example, in [43], the authors describe a method to systematically model service-oriented systems in a series of steps. The method aims at avoiding inconsistencies between models and has a natural order of steps. That is, constraints are imposed at language level before the actual system models are created. In [24], a six-step method for modelling software architecture is presented, composed of selecting an architectural style/pattern, defining architectural structural elements and rules, specifying data structures, specifying structural units, specifying mechanisms to support state models and timers, and eventually build the architecture model. Again, different models are used at different steps of the method, and the order of the steps is important. [35] describe guidelines for the use of OntoUML. These guidelines are essentially restrictions on what parts of the OntoUML notation to use. Neither of these three papers provides detailed guidance on *how* the individual models are created.

Finally, twelve papers mix guidance on what notations, diagrams, or formalisms to use with detailed instructions on how to use these. For example, [85] aims to improve the quality of information models by providing guidelines in the form of best practices. These best practices include instructions on *what* diagrams to create, and *how* to ensure several of them are consistent. [26] also provides guidelines, including *what* to use (e.g. in terms of tooling), and *how* to approach modelling of use cases.

4.3.3 View-based or perspective-based guidance

Another common way to provide modelling guidance, found in 6 out of 35 papers, is to propose modelling according to multiple different views or system concerns, similar to architectural documentation that is typically presented according to multiple dimensions or in multiple views [4].

For example, guidelines for increasing the quality of process models are presented in [5], using both perspectives and views. The six perspectives of correctness, relevance, economic efficiency, clarity, comparability, and systematic design are considered, as well as several modelling views, such as a data view, organisational view, and control view. Similarly, [38] proposes a method to semi-automatically create hypermedia applications by modelling the application using different views, such as the navigation and presentation views.

4.3.4 Decomposition-driven or refinement-driven guidance

A common approach to deal with complexity in computer science is to decompose a problem into sub-problems, or to incrementally refine a solution. These two approaches are also reflected in guidance on modelling. That is, several papers propose methods and guidelines in which an abstract, incomplete or informal model is incrementally refined.

For example, the method proposed in [59] suggests to incrementally formalise behaviour, starting from textual rules, which are then refined into graphical models, and finally formalised using colored Petri nets. Similarly, Marincic et al. [62] suggest a step-wise process in which requirements for embedded control systems are refined. A method to model cyber-physical systems in terms of abstraction levels is presented in [19], where abstract concepts such as initial requirements are modelled first, and then increasingly described in more detail. Ultimately, the method allows for simulation of the CPS' behaviour. A method that prescribes step-wise decomposition of business processes is described in [75]. Between each decomposition step, it is decided which parts should be expressed together in a process model. [65] describes decomposition steps, each of which is accompanied by guiding questions. Finally, in [11], a method to model agent-based systems is described, which relies on breaking down the overall process into modelling the agent space, the agent components, and the agents.

4.3.5 Best practices and anti-patterns

Best practices and anti-patterns are common ways to describe the state of practice and guide practitioners, e.g. in core computer science areas such as programming [56, 94] and software design [8]. This is also reflected in our data set, with 12 papers suggesting such best practices or anti-patterns.

For example, in [76], five modelling styles are proposed ranging from the so-called ultra light style, where UML activity diagrams are supported with free text, to the so-called precise operational style, where UML and OCL are used according to their semantics. While the authors call these *styles*, they are essentially best practices on which parts of the official UML notation should be used depending on the functional context (who, when, where, how and why will the model be used). The styles are based on the authors' understanding, without any empirical data supporting them. In another direction, Frank [28] provides general design guidelines for multi-level models. These include principles and rationales, e.g. specifying knowledge on the highest possible level to avoid redundancy.

4.3.6 Considering cognition

Finally, three papers in our dataset explicitly consider the modeller's cognitive processes, or argue that it should be taken into consideration.

In [86], the authors suggest that a mental model is created when a problem is conceptualised. The mental model is then mapped to the actual solution model. Assuming this process, the authors argue that improving the mental model could lead to an improved domain understanding. [14] argues that cognition needs to be taken into account to explain shortcomings in BPM. Finally, cognition is taken into account explicitly as a part of the theoretical framework in [18], where three styles for BPM are extracted based on an observational study. The results describe a "high-efficiency" style, a "good layout but less efficiency" style, and a "neither good layout nor very efficient" style. The authors describe task-specific and modeller-specific factors behind each style, especially how the above-mentioned two factors influence specific aspects of each style.

5 Discussion

The objective of our SLR is to investigate how much guidance exists in the literature for model creation in SE. We discuss these findings with respect to a number of observed themes below.

5.1 Little empirical evaluation

Among 35 included papers, 22 are solution proposals (see Table 2), and only 1 contains validation research. These statistics demonstrate a lack of empirical evaluation present in the literature. In contrast, the papers obtained through snowball sampling in BPM show a higher amount of evaluation and validation, with 4 out of 7 papers being evaluation research, and one paper being validation research. That is, the area of BPM seems to be more mature when it comes to supporting model creation through empirical studies. Similarly, the only paper explicitly considering cognition and the construction of mental models were in the area of BPM.

This lack of empirical studies is comparably common in the area of software modelling. Zhang et al. [101] find that empirical research methods within SE are mostly applied to software maintenance, quality and testing. While software models and methods gained empirical attention[3] but none of them investigating modelling style nor model creation.

5.2 Lack of categorisation and organisation

We used two search strings, one with direct keywords (**S1**) like "guidelines" and "creation"; other with generic terms (**S2**), such as "approach" and "method." From **S1**, we find a variety of papers that, according to our definitions, propose guidelines and methods. However, they use terms such as **guidelines**, **methods**, **frameworks** and others in a seemingly arbitrary fashion. We address this issue by proposing definitions that make use of existing ones as much as possible. Additionally, we outlined dimensions and properties of existing modelling guidelines and methods. Interestingly, the 6 papers collected through **S2** all propose methods according to our definition. This indicates that the terms used are not entirely arbitrary, as guidelines were only found in S1. Potentially, "methods" might subjectively be considered as something larger among researchers, and therefore commonly include stepwise instructions. Finally, we did not find any single paper in **S1** or **S2** that proposes a modelling style according to our definition.

Common ways to provide guidance in modelling are to describe what diagrams should be created, to prescribe views or perspectives that should be considered, to encourage a stepwise decomposition or refinement, and to suggest general best practices or anti-patterns. With the exception of stepwise decomposition/refinement, these forms of guidance can either follow a dedicated set of steps or not. For instance, modelling guidelines could simply outline a few UML diagrams that need to be created in arbitrary order, while a modelling method could prescribe the same diagrams, but in a stepwise fashion.

5.3 Limited consideration of cognition

In BPM, substantial work exists that explicitly discusses the role of cognitive processes and individual preferences in creating business process models. In the SE modelling community, we do not find such work outside of the BPM community. On the one hand, we believe this relates to the relatively specialised nature of BPM as opposed to system modelling on a general level. That is, notations, level of abstraction, and purpose vary considerably in software and systems modelling and are likely to affect the cognitive processes and modelling styles of individuals. Nevertheless, we consider it relevant to explore this direction more in the future. Existing work from the BPM community can be used as valuable guidance to design similar studies in software modelling as a whole. This work necessarily needs to include expertise from other fields, such as Psychology.

[3] 66 papers published from 2013–2017 at EMSE and ESEM.

6 Conclusion

We conducted an SLR on guidance in software modelling. Using two search strings (S1 and S2) and snowballing, we extracted 35 papers out of 80051 for full-text reading.

We find that terminologies are used arbitrary and inconsistently, and that little empirically validated modelling guidance exists. Existing papers use different dimensions to provide guidance for modellers, often in the form of prescribing which diagrams/notations to use, which views/perspectives to model, how to decompose or refine a model of a system, or by providing best practices or anti-patterns. In the BPM community, cognition has been explicitly considered as an important aspect of the modelling process. However, our results show that we lack similar work outside the BPM community.

Our results help researchers by highlighting gaps in research, e.g. on cognition in software modelling. The link to research in BPM clearly highlights differences in this direction, that could be picked up by modelling researchers in future work. Thus, we believe future work should explicitly consider cognition when creating models, and design guidance and tooling accordingly. This work should be multi-disciplinary, including researchers from, e.g. Psychology.Furthermore, to address the lack of consistent terminology, we provide definitions for modelling *guidelines*, *methods*, and *styles*. These definitions help to clearly articulate modelling guidance in future work. We believe this is an important step in order to ensure a greater clarity of research results in the community. One clear result from our study is that empirical validation is lacking. Hence, it is important to address this in future work and indeed validate proposed modelling guidance.

For modelling practitioners, our results show the possible forms modelling guidance can take. Thus, it serves as an inspiration on how engineers can be supported when creating and maintaining models for different purposes, e.g. for formal analyses, increased understandability of models, or communication among stakeholders.

A Paper categorisation details

In the following, we list the papers we used during our analysis. Table 7 shows the papers extracted during the original

Table 7 Selected papers from search_S1

Paper	Titles	Authors
[43]	Context-based modelling: Introducing a novel modelling approach	M. Juhrisch and G. Dietz
[59]	Scenario-based modelling in industrial information systems	R. J. Machado, J.o M. Fernandes, J. P. Barros and L. Gomes
[38]	A UML-Based Methodology for Hypermedia Design	R. Hennicker and N. Koch
[24]	Architecture modelling for translative model-driven development	A. Fatwanto and C. Boughton
	Modelling and simulation of CPS based on SysML and modellica(KG)	F. Deng, Y. Yan, F. Gao and Linbo Wu
	Modelling Industrial Embedded Systems with UML	J. M. Fernande, R. J. Machado and H. D. Santos
[76]	Business Process Modelling: Five Styles and a Method to Choose the Most Suitable One	G. Reggio, M. Leotta, F. Ricca and E. Astesiano
[35]	Design patterns and inductive modelling rules to support the construction of ontologically well-founded conceptual models in OntoUML	G. Guizzardi, A. P. das Graças, and R. S. S. Guizzardi
[31]	Guidelines for modelling reactive systems with coloured Petri nets	M. P. Gonçalves and J. M. Fernandes
[85]	The guidelines of modelling - An approach to enhance the quality in information models	R. Schuette and T. Rotthowe
[62]	Non-Monotonic Modelling from Initial Requirements: A Proposal and Comparison with Monotonic Modelling Methods	J. Marincic, A. Mader, H. Wupper and R. Wieringa
[18]	Model development guidelines for UML-RT: conventions, patterns and antipatterns	T. K. Das and J. Dingel
[102]	Design guidelines for feature model construction: Exploring the relationship between feature model structure and structural complexity	X. Zhao and J. Gray

Table 7 continued

Paper	Titles	Authors
[5]	Guidelines of business process modelling	J. Becker, M. Rosemann and C. van Uthmann
[95]	A modelling approach for use-cases model in UML	Z. Wang
[100]	A modelling methodology to facilitate safety-oriented architecture design of industrial avionics software	J. Wu, T. Yue, S. Ali and H. Zhang
[26]	Use case modelling guidelines	D. G. Firesmith
[44]	WWM: a practical methodology for Web application modelling	C. Kaewkasi and W. Rivepiboon
[17]	A BPMN-driven framework for Multi-Robot System development	F. Corradini, S. Pettinari, B. Re, L. Rossi, and F. Tiezzi
[32]	MDD4CPD: Model-Driven Development Approach Proposal for Cyber-Physical Devices	R. F. Goncalves, A. Menolli and G. M. Dionisio
[37]	Communication Oriented Modelling of Evolving Systems of Systems	S. K. R. Harbo, M. K. Kristensen, E. P. Voldby, S. V. Andersen, F. C. Petersen and M. Albano
[28]	Prolegomena of a Multi-Level Modelling Method Illustrated with the FMML x	U. Frank

Table 8 Selected papers from snowballing set

Paper	Titles	Authors
[86]	Towards Understanding the Process of Process Modelling: Theoretical and Empirical Considerations	P. Soffer, M. Kaner and Y. Wand
	Guiding goal modelling with scenarios	C. Rolland, C. Souveyet and C. Ben Achour
[65]	Modelling Families of Business Process Variants: A Decomposition Driven Method	F. Milani, M. D., N. Ahmed and R. Matulevičius
[16]	Guidelines of a Unified Approach for Product and Business Process Modelling in Complex Enterprise	A. Corallo, P. De Paolis, M. Ippoliti, M. Lazoi, M. Scalvenzi and G. Secundo
[75]	Modelling Styles in Business Process Modelling	J. Pinggera, P. Soffer, S. Zugal, B. Weber, M. Weidlich, D. Fahland, H. A. Reijers and J. Mendling
[14]	The Structured Process Modelling Theory (SPMT) a cognitive view on why and how modellers benefit from structuring the process of process modelling	J. Claes, I. Vanderfeesten, F. Gailly, P. Grefen and G. Poels
[63]	Seven Process Modelling Guidelines (7PMG)	J. Mendling, H. A. Reijers and W. M. P. van der Aalst

Table 9 Selected papers from search_S2

Paper	Titles	Authors
[103]	Multidisciplinary interface model for design of mechatronic systems	C. Zheng, J. Le Duigou, M. Bricogne and B. Eynard
[58]	Model-based security engineering for secure systems development	A. Lunkeit and H. Pohl
[88]	User interface derivation from business processes: a model-driven approach for organisational engineering	K. Sousa, H. Mendonça, J. Vanderdonckt, E. Rogier and J. Vandermeulen
[11]	Supporting agent-based distributed software development through modelling and simulation	L. Cai, CK Chang and J. Cleland-Huang
[34]	Towards a Human-Centered UML for Risk Analysis: Application to a medical robot	J. Guiochet, G. Motet, C. Baron and G. Boy
[66]	Integrating System and Software Engineering Through Modelling	J. Mindock and G. Watney

search, while Table 8 shows the papers from the snowball search.

References

1. Abbott, R.J.: Program design by informal English descriptions. Commun. ACM **26**(11), 882–894 (1983). https://doi.org/10.1145/182.358441
2. Allinson, C., Hayes, J.: The cognitive style index: a measure of intuition analysis for organizational research. J. Manag. Stud. **33**, 119–135 (1996). https://doi.org/10.1111/j.1467-6486.1996.tb00801.x
3. Autili, M., Grunske, L., Lumpe, M., Pelliccione, P., Tang, A.: Aligning qualitative, real-time, and probabilistic property specification patterns using a structured English grammar. IEEE Trans. Softw. Eng. **41**(7), 620–638 (2015). https://doi.org/10.1109/TSE.2015.2398877
4. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional (2003)
5. Becker, J., Rosemann, M., Von Uthmann, C.: Guidelines of business process modeling. In: Business Process Management, pp. 30–49. Springer (2000)
6. Bézivin, J., Muller, P. (eds.): The Unified Modeling Language, UML'98: Beyond the Notation, First International Workshop, Mulhouse, France, June 3-4, 1998, Selected Papers, Lecture Notes in Computer Science, vol. 1618. Springer (1999). https://doi.org/10.1007/b72309
7. Bordeleau, F.: A Systematic and Traceable Progression from Scenario Models to Communicating Hierarchical State Machines. Ph.D. thesis. Carleton University (2000)
8. Brown, W.H., Malveau, R.C., McCormick, H.W.S., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley & Sons, Inc. (1998)
9. Bruegge, B., Dutoit, A.H.: Object-Oriented Software Engineering Using UML, Patterns, and Java, 3rd edn. Prentice Hall Press (2009)
10. Burgueño, L., Cabot, J., Wimmer, M., Zschaler, S.: Guest editorial to the theme section on ai-enhanced model-driven engineering. Softw. Syst. Model. **21**(3), 963–965 (2022)
11. Cai, L., Chang, C.K., Cleland-Huang, J.: Supporting agent-based distributed software development through modeling and simulation. In: The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems, 2003. FTDCS 2003. Proceedings, pp. 56–62. IEEE (2003)
12. Cámara, J., Troya, J., Burgueño, L., Vallecillo, A.: On the assessment of generative ai in modeling tasks: an experience report with chatgpt and uml. Softw. Syst. Model. 1–13 (2023)
13. Chaaben, M.B., Burgueño, L., Sahraoui, H.: Towards using few-shot prompt learning for automating model completion. arXiv preprint arXiv:2212.03404 (2022)
14. Claes, J., Vanderfeesten, I., Gailly, F., Grefen, P., Poels, G.: The structured process modeling theory (spmt) a cognitive view on why and how modelers benefit from structuring the process of process modeling. Inf. Syst. Front. **17**, 1401–1425 (2015). https://doi.org/10.1007/s10796-015-9585-y
15. Claes, J., Vanderfeesten, I., Pinggera, J., Reijers, H.A., Weber, B., Poels, G.: A visual analysis of the process of process modeling. Inf. Syst. e-Bus. Manag. **13**(1), 147–190 (2015)
16. Corallo, A., Paolis, P., Ippoliti, M., Lazoi, M., Scalvenzi, M., Secundo, G.: Guidelines of a unified approach for product and business process modeling in complex enterprise. Knowl. Process Manag. (2011). https://doi.org/10.1002/kpm.381
17. Corradini, F., Pettinari, S., Re, B., Rossi, L., Tiezzi, F.: A bpmn-driven framework for multi-robot system development. Robot. Auton. Syst. **160**, 104,322 (2023)
18. Das, T., Dingel, J.: Model development guidelines for uml-rt: conventions, patterns and antipatterns. Softw. Syst. Model. (2018). https://doi.org/10.1007/s10270-016-0549-6
19. Deng, F., Yan, Y., Gao, F., Wu, L.: Modeling and simulation of cps based on sysml and modelica (kg). In: Proceedings of the 31st International Conference on Software Engineering & Knowledge Engineering SEKE 2019 (2019)
20. Douglass, B.P.: Doing hard time: developing real-time systems with UML, objects, frameworks, and patterns, vol. 1. Addison-Wesley Professional (1999)
21. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Property specification patterns for finite-state verification. In: Proceedings of the Second Workshop on Formal Methods in Software Practice, pp. 7–15. ACM (1998)
22. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No. 99CB37002), pp. 411–420. IEEE (1999)
23. Fang, J., Zhu, Z., Li, S., Su, H., Yu, Y., Zhou, J., You, Y.: Parallel training of pre-trained models via chunk-based dynamic memory management. IEEE Trans. Parallel Distrib. Syst. **34**(1), 304–315 (2022)
24. Fatwanto, A., Boughton, C.: Architecture modeling for translative model-driven development. In: 2008 International Symposium on Information Technology, vol. 1, pp. 1–9 (2008). https://doi.org/10.1109/ITSIM.2008.4631619
25. Fernandes, J., Machado, R., Santos, H.: Modeling industrial embedded systems with uml. In: Proceedings of the Eighth International Workshop on Hardware/Software Codesign. CODES 2000, pp. 18–22 (2000). https://doi.org/10.1109/HSC.2000.843700
26. Firesmith, D.: Use case modeling guidelines. In: Proceedings of Technology of Object-Oriented Languages and Systems - TOOLS 30 (Cat. No.PR00278), pp. 184–193 (1999). https://doi.org/10.1109/TOOLS.1999.787548
27. Fleiss, J.L., Levin, B., Paik, M.C.: Statistical Methods for Rates and Proportions. Wiley (2013)
28. Frank, U.: Prolegomena of a multi-level modeling method illustrated with the fmml x. In: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 521–530. IEEE (2021)
29. Gamma, E.: Design patterns: elements of reusable object-oriented software. Pearson Education India (1995)
30. Giraldo, F.D., España, S., Pastor, O.: Analysing the concept of quality in model-driven engineering literature: a systematic review. In: 2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS), pp. 1–12. IEEE (2014)
31. Gonçalves, M., Fernandes, J.M.: Guidelines for modelling reactive systems with coloured petri nets. In: Machado, R.J., Maciel, R.S.P., Rubin, J., Botterweck, G. (eds.) Model-Based Methodologies for Pervasive and Embedded Software, pp. 126–137. Springer, Berlin Heidelberg, Berlin, Heidelberg (2013)
32. Goncalves, R.F., Menolli, A., Dionisio, G.M.: Mdd4cpd: model driven development approach proposal for cyber-physical devices. In: Anais do XVIII Simpósio Brasileiro de Sistemas de Informação. SBC (2022)
33. Grunske, L.: Specification patterns for probabilistic quality properties. In: 2008 ACM/IEEE 30th International Conference on Software Engineering, pp. 31–40. IEEE (2008)
34. Guiochet, J., Motet, G., Baron, C., Boy, G.: Toward a human-centered uml for risk analysis: Application to a medical robot. In: Human Error, Safety and Systems Development: IFIP 18th World

Computer Congress TC13/WC13. 5 7th Working Conference on Human Error, Safety and Systems Development 22–27 August 2004 Toulouse, France, pp. 177–191. Springer (2004)
35. Guizzardi, G., das Graças, A.P., Guizzardi, R.S.S.: Design patterns and inductive modeling rules to support the construction of ontologically well-founded conceptual models in ontouml. In: C. Salinesi, O. Pastor (eds.) Advanced Information Systems Engineering Workshops, pp. 402–413. Springer, Berlin, Heidelberg (2011)
36. Gwet, K.L.: Handbook of Inter-rater Reliability Advanced Analytics. LLC, Gaithersburg, MD (2010)
37. Harbo, S.K.R., Kristensen, M.K., Voldby, E.P., Andersen, S.V., Petersen, F.C., Albano, M.: Communication oriented modeling of evolving systems of systems. In: 2021 16th International Conference of System of Systems Engineering (SoSE), pp. 88–94. IEEE (2021)
38. Hennicker, R., Koch, N.: A uml-based methodology for hypermedia design. In: Evans, A., Kent, S., Selic, B. (eds.) ≪ UML ≫ 2000 – The Unified Modeling Language, pp. 410–424. Springer, Berlin Heidelberg, Berlin, Heidelberg (2000)
39. Hutchinson, J., Rouncefield, M., Whittle, J.: Model-driven engineering practices in industry. In: 33rd International Conference on Software Engineering (ICSE '11), pp. 633–642 (2011)
40. Hutchinson, J., Whittle, J., Rouncefield, M.: Model-driven engineering practices in industry: social, organizational and managerial factors that lead to success or failure. Sci. Comput. Program. **89**(Part B), 144–161 (2014)
41. Hutchinson, J., Whittle, J., Rouncefield, M., Kristoffersen, S.: Empirical assessment of MDE in industry. In: 33rd International Conference on Software Engineering (ICSE '11), pp. 471–480 (2011)
42. Isaksen, S., Kaufmann, A., Bakken, B.T.: An examination of the personality constructs underlying dimensions of creative problem-solving style. J. Creat. Behavi. **50**, 268–281 (2016)
43. Juhrisch, M., Dietz, G.: Context-based modeling: introducing a novel modeling approach. In: Esswein, W., Turowski, K., Juhrisch, M. (eds.) Modellierung betrieblicher Informationssysteme (MobIS 2010). Modellgestütztes Management, pp. 111–130. Gesellschaft für Informatik e.V., Bonn (2010)
44. Kaewkasi, C., Rivepiboon, W.: Wwm: a practical methodology for web application modeling. In: Proceedings 26th Annual International Computer Software and Applications, pp. 603–608 (2002). https://doi.org/10.1109/CMPSAC.2002.1045070
45. Kharchenko, V., Fesenko, H., Illiashenko, O.: Quality models for artificial intelligence systems: characteristic-based approach, development and application. Sensors **22**(13), 4865 (2022)
46. Kılıç, S.: Kappa testi. J. Mood Disord. **5**(3) (2015)
47. Kitchenham, B., Brereton, P., Budgen, D., Turner, M., Bailey, J., Linkman, S.: Systematic literature reviews in software engineering-a systematic literature review. Inf. Softw. Technol. **51**, 7–15 (2009). https://doi.org/10.1016/j.infsof.2008.09.009
48. Kitchenham, B.A., Charters, S.: Guidelines for performing systematic literature reviews in software engineering. Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report (2007)
49. Kuhrmann, M., Méndez Fernández, D., Daneva, M.: On the pragmatic design of literature studies in software engineering: an experience-based guideline. Empir. Softw. Eng. **22**, 2852–2891 (2017). https://doi.org/10.1007/s10664-016-9492-y
50. Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. Biometrics 159–174 (1977)
51. Langford, M.A., Chan, K.H., Fleck, J.E., McKinley, P.K., Cheng, B.H.: Modalas: model-driven assurance for learning-enabled autonomous systems. In: 2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 182–193. IEEE (2021)
52. Liebel, G.: Model-Based Requirements Engineering in the Automotive Industry: Challenges and Opportunities. Chalmers Tekniska Högskola (Sweden) (2016)
53. Liebel, G., Marko, N., Tichy, M., Leitner, A., Hansson, J.: Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Softw. Syst. Model. **17**(1), 91–113 (2018). https://doi.org/10.1007/s10270-016-0523-3
54. Liebel, G., Tichy, M., Knauss, E.: Use, potential, and showstoppers of models in automotive requirements engineering. Softw. Syst. Model. (2018). https://doi.org/10.1007/s10270-018-0683-4
55. Lohmeyer, Q., Meboldt, M., et al.: How we understand engineering drawings: an eye tracking study investigating skimming and scrutinizing sequences. In: International conference on engineering design ICED, vol. 15 (2015)
56. Long, F., Mohindra, D., Seacord, R.C., Sutherland, D.F., Svoboda, D.: Java Coding Guidelines: 75 Recommendations for Reliable and Secure Programs. Addison-Wesley (2013)
57. Loniewski, G., Insfran, E., Abrahão, S.: A systematic review of the use of requirements engineering techniques in model-driven development. In: International Conference on Model Driven Engineering Languages and Systems, pp. 213–227. Springer (2010)
58. Lunkeit, A., Pohl, H.: Model-based security engineering for secure systems development. In: ARCS Workshop 2018; 31th International Conference on Architecture of Computing Systems, pp. 1–10. VDE (2018)
59. Machado, R.J., Fernandes, J.M., Barros, J.P., Gomes, L.: Scenario-based modeling in industrial information systems. In: Hinchey, M., Kleinjohann, B., Kleinjohann, L., Lindsay, P.A., Rammig, F.J., Timmis, J., Wolf, M. (eds.) Distributed, Parallel and Biologically Inspired Systems, pp. 19–30. Springer, Berlin Heidelberg, Berlin, Heidelberg (2010)
60. Maier, A., Baltsen, N., Christoffersen, H., Störrle, H.: Towards diagram understanding: a pilot study measuring cognitive workload through eye-tracking. In: Proceedings of International Conference on Human Behaviour in Design 2014 (2014)
61. Maoz, S., Ringert, J.O.: Gr(1) synthesis for ltl specification patterns. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pp. 96–106. Association for Computing Machinery (2015). https://doi.org/10.1145/2786805.2786824
62. Marincic, J., Mader, A., Wupper, H., Wupper, H., Wieringa, R.: Non-monotonic modelling from initial requirements: a proposal and comparison with monotonic modelling methods. In: Proceedings of the 3rd International Workshop on Applications and Advances of Problem Frames, pp. 67–73 (2008). https://doi.org/10.1145/1370811.1370825
63. Mendling, J., Reijers, H., Aalst, W.: Seven process modeling guidelines (7pmg). Inf. Softw. Technol. **52**, 127–136 (2010). https://doi.org/10.1016/j.infsof.2009.08.004
64. Messick, S.: The nature of cognitive styles: problems and promise in educational practice. Educ. Psychol. **19**, 59–74 (1984)
65. Milani, F., Dumas, M., Ahmed, N., Matulevičius, R.: Modelling families of business process variants: a decomposition driven method. Inf. Syst. (2013). https://doi.org/10.1016/j.is.2015.09.003
66. Mindock, J., Watney, G.: Integrating system and software engineering through modeling. In: 2008 IEEE Aerospace Conference, pp. 1–12. IEEE (2008)
67. Mohagheghi, P., Dehlen, V.: Where is the proof?—A review of experiences from applying mde in industry. In: Schieferdecker, I., Hartman, A. (eds.) Model Driven Architecture—Foundations and Applications. Lecture Notes in Computer Science, vol. 5095, pp. 432–443. Springer, Berlin Heidelberg (2008)
68. Mohagheghi, P., Gilani, W., Stefanescu, A., Fernandez, M.A., Nordmoen, B., Fritzsche, M.: Where does model-driven engineer-

ing help? experiences from three industrial cases. Softw. Syst. Model. **12**(3), 619–639 (2013)
69. Moody, D.: The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. IEEE Trans. Softw. Eng. **35**(6), 756–779 (2009). https://doi.org/10.1109/TSE.2009.67
70. Nguyen, P.H., Klein, J., Le Traon, Y., Kramer, M.E.: A systematic review of model-driven security. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), vol. 1, pp. 432–441. IEEE (2013)
71. Nguyen, P.H., Kramer, M., Klein, J., Le Traon, Y.: An extensive systematic review on the model-driven development of secure systems. Inf. Softw. Technol. **68**, 62–81 (2015)
72. Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering, p. 17 (2008)
73. Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: an update. Inf. Softw. Technol. **64**, 1–18 (2015). https://doi.org/10.1016/j.infsof.2015.03.007
74. Pinggera, J., Soffer, P., Fahland, D., Weidlich, M., Zugal, S., Weber, B., Reijers, H., Mendling, J.: Styles in business process modeling: an exploration and a model. Softw. Syst. Model. (2013). https://doi.org/10.1007/s10270-013-0349-1
75. Pinggera, J., Soffer, P., Zugal, S., Weber, B., Weidlich, M., Fahland, D., Reijers, H.A., Mendling, J.: Modeling styles in business process modeling. In: Bider, I., Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., Wrycza, S. (eds.) Enterprise, Business-Process and Information Systems Modeling, pp. 151–166. Springer, Berlin Heidelberg, Berlin, Heidelberg (2012)
76. Reggio, G., Leotta, M., Ricca, F., Astesiano, E.: Business process modelling: five styles and a method to choose the most suitable one. In: Proceedings of the Second Edition of the International Workshop on Experiences and Empirical Studies in Software Modelling, EESSMod '12. Association for Computing Machinery, New York, NY, USA (2012). https://doi.org/10.1145/2424563.2424574
77. Rivera, L.F., Müller, H.A., Villegas, N.M., Tamura, G., Jiménez, M.: On the engineering of iot-intensive digital twin software systems. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp. 631–638 (2020)
78. Rolland, C., Souveyet, C., Achour, C.: Guiding goal modeling using scenarios. IEEE Trans. Softw. Eng. TSE (1999). https://doi.org/10.1109/32.738339
79. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., Lorensen, W.E., et al.: Object-Oriented Modeling and Design, vol. 199. Prentice-hall Englewood Cliffs, NJ (1991)
80. Runeson, P., Höst, M., Rainer, A., Regnell, B.: Case study research in software engineering—guidelines and examples (2012)
81. Saini, R., Mussbacher, G., Guo, J.L., Kienzle, J.: Automated, interactive, and traceable domain modelling empowered by artificial intelligence. Softw. Syst. Model. 1–31 (2022)
82. Santos, M., Gralha, C., Goulão, M., Araújo, J.: Increasing the semantic transparency of the kaos goal model concrete syntax. In: Trujillo, J.C., Davis, K.C., Du, X., Li, Z., Ling, T.W., Li, G., Lee, M.L. (eds.) Conceptual Modeling, pp. 424–439 (2018)
83. Santos, M., Gralha, C., Goulão, M., Araújo, J., Moreira, A.: On the impact of semantic transparency on understanding and reviewing social goal models. In: 2018 IEEE 26th International Requirements Engineering Conference (RE), pp. 228–239 (2018). https://doi.org/10.1109/RE.2018.00031
84. Schätz, B., Törngreen, M., Bensalem, S., Cengarle, M.V., Pfeifer, H., McDermid, J., Passerone, R., Sangiovanni-Vincentelli, A.L.: Cyber-physical european roadmap and strategy: research agenda and recommendations for action. CyPhERS . Tech. Rep (2015)
85. Schuette, R., Rotthowe, T.: The guidelines of modeling – an approach to enhance the quality in information models. In: Ling, T.W., Ram, S., Li Lee, M. (eds.) Conceptual Modeling – ER '98, pp. 240–254. Springer Berlin Heidelberg, Berlin, Heidelberg (1998)
86. Soffer, P., Kaner, M., Wand, Y.: Towards understanding the process of process modeling: theoretical and empirical considerations. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) Business Process Management Workshops, pp. 357–369. Springer, Berlin, Heidelberg (2012)
87. Somogyi, F.A., Asztalos, M.: Systematic review of matching techniques used in model-driven methodologies. Softw. Syst. Model. **19**(3), 693–720 (2020)
88. Sousa, K., Mendonça, H., Vanderdonckt, J., Rogier, E., Vandermeulen, J.: User interface derivation from business processes: a model-driven approach for organizational engineering. In: Proceedings of the 2008 ACM symposium on Applied computing, pp. 553–560 (2008)
89. Störrle, H.: On the impact of layout quality to understanding uml diagrams: Size matters. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Insfran, E. (eds.) Model-Driven Engineering Languages and Systems, pp. 518–534 (2014)
90. Störrle, H.: Diagram size vs. layout flaws: understanding quality factors of uml diagrams. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16, pp. 31:1–31:10 (2016)
91. Störrle, H.: On the impact of size to the understanding of uml diagrams. Softw. Syst. Model. **17**(1), 115–134 (2018). https://doi.org/10.1007/s10270-016-0529-x
92. Störrle, H., Fish, A.: Towards an operationalization of the "physics of notations" for the analysis of visual languages. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) Model-Driven Engineering Languages and Systems, pp. 104–120. Springer, Berlin, Heidelberg (2013)
93. Sunkle, S., Saxena, K., Patil, A., Kulkarni, V.: Ai-driven streamlined modeling: experiences and lessons learned from multiple domains. Softw. Syst. Model. **21**(3), 1–23 (2022)
94. Sutter, H., Alexandrescu, A.: C++ coding standards: 101 rules, guidelines, and best practices. Pearson Education (2004)
95. Wang, Z.: A modeling approach for use-cases model in uml. In: 2012 IEEE Fifth International Conference on Advanced Computational Intelligence (ICACI), pp. 176–179 (2012). https://doi.org/10.1109/ICACI.2012.6463145
96. Weyssow, M., Sahraoui, H., Syriani, E.: Recommending metamodel concepts during modeling activities with pre-trained language models. Softw. Syst. Model. **21**(3), 1071–1089 (2022)
97. Whittle, J., Hutchinson, J., Rouncefield, M., Burden, H., Heldal, R.: Industrial adoption of model-driven engineering: Are the tools really the problem? In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P. (eds.) Model-Driven Engineering Languages and Systems. Lecture Notes in Computer Science, vol. 8107, pp. 1–17. Springer, Berlin Heidelberg (2013)
98. Wieringa, R., Maiden, N., Mead, N., Rolland, C.: Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. **11**, 102–107 (2006). https://doi.org/10.1007/s00766-005-0021-6
99. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, EASE '14. Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2601248.2601268
100. Wu, J., Yue, T., Ali, S., Zhang, H.: A modeling methodology to facilitate safety-oriented architecture design of industrial avionics

software. Softw. Pract. Exp. (2015). https://doi.org/10.1002/spe.2281
101. Zhang, L., Tian, J.H., Jiang, J., Liu, Y.J., Pu, M.Y., Yue, T.: Empirical research in software engineering–a literature survey. J. Comput. Sci. Technol. (2018). https://doi.org/10.1007/s11390-018-1864-x
102. Zhao, X., Gray., J.: Design guidelines for feature model construction: Exploring the relationship between feature model structure and structural complexity. In: Proceedings of the 7th International Conference on Model-Driven Engineering and Software Development - MODELSWARD, pp. 325–333. INSTICC, SciTePress (2019). https://doi.org/10.5220/0007388703250333
103. Zheng, C., Le Duigou, J., Bricogne, M., Eynard, B.: Multidisciplinary interface model for design of mechatronic systems. Comput. Ind. **76**, 24–37 (2016)

Shalini Chakraborty completed her bachelor's (2017) and master's (2019) in Computer Science from Calcutta University, India. She is pursuing her Ph.D., working in Software Engineering (SE), primarily focusing on Model-Based Engineering (MBE) and empirical research. She is interested in human factors in SE and social software engineering. Currently, she is employed at Reykjavik University as a Ph.D. candidate and teaching assistant.

Grischa Liebel is an Assistant Professor in Software Engineering and the director of the CRESS research centre at Reykjavik University, Iceland. He holds a PhD degree from Chalmers University, Sweden. His research focuses on human factors in software engineering, typically in areas such Modelling and Model-Based Engineering, Requirements Engineering, Processes, and Education. Much of Grischa's research is done in collaboration with industry (and with humans).

Chapter 7

Paper II: We do not understand what it says–studying student perceptions of software modelling

Shalini Chakraborty, Grischa Liebel
Empirical Software Engineering (EMSE) 28 (2023), 149
https://doi.org/10.1007/s10664-023-10404-w

Empirical Software Engineering (2023) 28:149
https://doi.org/10.1007/s10664-023-10404-w

We do not understand what it says – studying student perceptions of software modelling

Shalini Chakraborty[1] · **Grischa Liebel**[1]

Accepted: 3 October 2023 / Published online: 9 November 2023

Abstract

Background Despite the potential benefits of software modelling, developers have shown a considerable reluctance towards its application. There is substantial existing research studying industrial use and technical challenges of modelling. However, there is a lack of detailed empirical work investigating how students perceive modelling.

Aim We investigate the perceptions of students towards modelling in a university environment.

Method We conducted a multiple case study with 5 cases (5 courses from 3 universities) and two units of analysis (student and instructor). We collected data through 21 semi-structured interviews, which we analysed using in-vivo coding and thematic analysis.

Results Students see some benefits of modelling, e.g., using models for planning and communicating within the group. However, several factors negatively influence their understanding of modelling, e.g., assignments with unclear expectations, irregular and insufficient feedback on their models, and lack of experience with the problem domains.

Conclusions Our findings help in understanding better why students struggle with software modelling, and might be reluctant to adopt it later on. Our recommendations on modelling education could help to improve education and training in software modelling, both at university and in industry. Specifically, we recommend that educators try to provide feedback beyond syntactical issues, and to consider using problem domains that students are knowledgeable about.

Keywords Software modelling · Case study · UML · Education

1 Introduction

Software modelling has the potential to improve on several factors in software and systems engineering, such as productivity (Agner et al. 2013) or cost (Kirstan and Zimmermann

Communicated by: Marco Torchiano

Grischa Liebel
grischal@ru.is

Shalini Chakraborty
shalini19@ru.is

[1] School of Technology, Department of Computer Science, Reykjavik University, Menntavegur 1, 102 Reykjavík, Iceland

2010). While these benefits seem to be appreciated in certain areas of software and systems engineering, like the use of UML diagrams (Anda et al. 2006; Dobing and Parsons 2008), software modelling is not widely adopted in software and systems engineering as a whole (Gorschek et al. 2014). Significantly, the benefits of software modelling in the embedded system domain are largely unexplored (Liebel et al. 2018a).

The reasons for a lack of adoption and use have been studied in depth, revealing issues such as poor quality code generation, lack of tool support, and lack of guidance or training (Forward et al. 2010; Liebel et al. 2018a, b; Mohagheghi et al. 2013; Whittle et al. 2013, 2014). Additionally, it has been put forward that a perceived lack of usefulness or too high effort prevents engineers from using models (Gorschek et al. 2014; Torchiano et al. 2013).

In the educational domain, substantial work exists on how to teach modelling in a university curriculum, e.g., Burgueño et al. (2018); Kolovos and Cabot (2016); Liebel et al. (2016); Paige et al. (2014); Stikkolorum et al. (2015); Westphal (2019); Whittle and Hutchinson (2011). Existing work typically comes in the form of suggested course designs, e.g., Schmidt et al. (2014); Westphal (2019), experience reports that discuss challenges or good practices, especially tool-related, e.g., Burgueño et al. (2018); Kolovos and Cabot (2016); Paige et al. (2014); Whittle and Hutchinson (2011), or papers that report quantitative studies of student opinions, e.g., Liebel et al. (2016); Stikkolorum et al. (2015). This body of work forms a rich source of experience and inspiration for teaching modelling.

However, there is a lack of in-depth studies that aim to understand how students perceive modelling and why this is the case. Experience reports commonly suffer from various biases, as they rely on individual experiences and lack rigorous methods of data collection and analysis. Similarly, existing quantitative studies aim to provide a broad picture and can, therefore, not provide detailed explanations (Stol and Fitzgerald 2018). Finally, experience reports by researchers that have a particular focus on software and systems modelling risk being unrepresentative of instructors without such a focus. Since students will after graduation become professionals, it is important to understand their perceptions. Understanding existing challenges they face, but also perceived benefits of using models can help improve modelling education and facilitate an increased uptake in industry.

To address this gap, this paper aims to investigate students' perceptions of modelling, both in terms of benefits of and challenges with modelling. We pose the following two research questions (RQs) to address our aim:

- RQ1: What are students' perceptions about modelling?
- RQ2: What are the main challenges students face while modelling?

To answer these questions, we conducted a qualitative case study, interviewing a total of 21 participants from 5 university courses that cover modelling. We interviewed both students and instructors, and analysed course assignments.

Our findings show that several of the perceived benefits and challenges align well with those reported in existing literature. Students find it beneficial to use models as a means of communication and handling complexity. They report difficulties such as obtaining good and fast feedback on a model. We further find that the lack of technical skills and domain knowledge in the student population prevents them from connecting their models and judging their quality to any known domain. This finding gives a better explanation of why certain problem domains might be problematic in any given context. Further, it highlights the importance to carefully tailor modelling courses to the experience of the audience, and consider the role of modelling courses in the curriculum.

The rest of the paper is structured as follows. In Section 2 we discuss the related work on modelling in education and industry, and the importance of taking students' perception into account. In Section 3, we present the method we applied for our case study, followed by the results in Section 4 and a discussion thereof in Section 5. Finally we conclude the paper in Section 6 with a summary and plans for future work.

2 Related Work

In the following, we describe related work that investigates the use of models in industry, and work on modelling education. Finally, we describe work that aims to connect industry and education.

2.1 Models in Industry

There is substantial work on the adoption and the challenges of software modelling in industry (Baker et al. 2005; Hutchinson et al. 2011a, b, (2014); Liebel et al. 2018a, b; Mohagheghi and Dehlen 2008; Mohagheghi et al. 2013; Torchiano et al. 2013; Whittle et al. 2014).

In an early study at Motorola, Baker et al. (2005) find that models increase productivity and reduce defects. However, they also find that modelling tools are insufficient and lack interoperability, and that MBE does not scale sufficiently.

Mohagheghi and Dehlen (2008); Mohagheghi et al. (2013) highlight the potential of using models for simulation and testing, while also mentioning tool issues and model complexity as challenges of adopting MBE.

Hutchinson et al. (2011a, b, (2014)); Whittle et al. (2014) conduct several studies assessing the state of practice of MBE (Model Based Engineering) in industry. The authors confirm that modelling tools and model complexity are problematic, but also highlight organisational factors. Among others, they find that a lack of training hinders the adoption of modelling.

In a survey in the Italian software developing industry, Torchiano et al. (2013) find that developers mainly use models for informal sketches and communication. They further report that inexperience among developers limits the use of models.

In a survey among systems engineering professionals, we find that models provide substantial benefits, e.g., in terms of increases in productivity (Liebel et al. 2018a). However, several challenges in technical and non-technical areas remain, e.g., lack of training and guidance, tool shortcomings, and a lack of tool interoperability. Following up with an in-depth qualitative study at two automotive companies, we report that models are used primarily for communication and to handle complexity. Stakeholders further prefer sketches and informal models (Liebel et al. 2018b). System models are often used in other domains, such as healthcare, for communication and to provide better organisational support. In Kopach-Konrad et al. (2007), authors discuss the application of event-based models such as Petri nets to capture the various operations running in a hospital. Although found helpful, the authors also highlight certain challenges like needing more understanding in adopting the models from healthcare associations. Considering UML specifically, the healthcare domain has multiple evidence of using UML models to design the domains (Raistrick 2004; Walderhaug et al. 2008) and human-centric tasks (Bernonville et al. 2005). However, in the latter, the authors mentioned a need for more explicit representation from the models. In the healthcare domain where the cognitive behaviour of the actors is highly involved in the process, software models that are used for designing those processes should provide the ability to include the behaviours.

2.2 Models in Education

Research on models in education exists primarily in three forms: (i) solution proposals on how to teach modelling, including tools tailored to an educational context, (ii) experience reports on modelling education, and (iii) surveys among students.

An example of the first category is the practical approach to teaching model-driven software development proposed by Schmidt et al. (2014). In the proposed course, students develop a code generator using standard software development tools. Similarly, Westphal (2019) describes the design of an SE course that focuses heavily on modelling. In Gilson (2018), authors proposed a course where they used students both as language designers and its users to evaluate the usability of software language engineering (SLE). Gonnord and Mosser (2018) try a reverse approach, and with the students, they journey from low-level C code to designing a modelling language workbench. In all cases, evaluation of the proposed course design relies on student feedback.

Numerous experience reports exist related to modelling education (Akayama et al. 2013; Kolovos and Cabot 2016; Paige et al. 2014). Akayama et al. (2013) share their experience and opinion on tool use in software modelling education. The paper describes different approaches taken by the individual authors and provides a discussion of factors such as modelling tools vs pen and paper, the conflict between the concepts of design and programming, and how to measure the quality of models. Paige et al. (2014) discuss what they consider to be bad practices of teaching modelling. They name bad practices such as covering a too broad range of modelling-related topics, and focusing on syntax instead of semantics. Similarly, Kolovos and Cabot (2016) present a corpus of use cases for courses teaching MBE. The authors state that modelling courses regularly suffer from the use of uninteresting or irrelevant examples, and therefore propose a list of use cases suited to teach modelling. Moody (2010) mentions a need for more effort in designing visual syntax for notations. Therefore, he proposes a tutorial that defines a "set of principles for designing cognitively effective visual notations: ones that are optimised for human communication and problem solving". Ciccozzi et al. (2018) present a survey among educators on how modelling is taught. The authors find that educators see the focus on tools critical, as it might prevent students from understanding the core principles of modelling.

Several empirical studies exist on modelling education (Agner et al. 2019; Hammouda et al. 2014; Liebel et al. 2017; Reuter et al. 2020; Stikkolorum et al. 2015). Reuter et al. (2020) study students' problems with UML diagrams over two modelling courses. As a result, the authors present a catalogue of problems with UML diagrams. In a similar direction, Stikkolorum et al. (2015) study student problems and modelling strategies in UML Class diagrams by presenting students with a specialised UML editor that incorporates feedback mechanisms. The results reveal four distinct strategies (depth-less, depth-first, breadth-first and ad hoc) which are four different ways to draw a class diagram for a given problem. These strategies are based on how they draw the class diagram with necessary associations and attributes in the given task and a number of problems, such as choosing the right syntax elements. Agner et al. (2019) conduct a survey on modelling tool use among 117 students. The authors report issues such as a lack of feedback, difficulties in drawing the diagrams and tool complexity. Hammouda et al. (2014) compare the use of modelling tools to pen and paper use. Using a survey, they evaluate students' perceptions of the differences, finding no clear advantage for either approach. In the context of modelling tools in education, the tool Umple needs to be highlighted (Garzón et al. 2015). Umple is a tool that allows to create UML models in a textual concrete syntax close to object-oriented languages like Java. Furthermore, code in many different general-purpose programmming languages can generated directly from

Umple models, thus allowing for direct feedback from models. Surveys with students have shown positive results on the use of Umple (Lethbridge et al. 2011; Liebel et al. 2017). Umple is successfully used by several instructors teaching modelling. However, to our knowledge, it is currently not widely used on a global scale.

Finally, we conducted several case studies on tool use in modelling education (Liebel et al. 2016, 2017). Tools like Umple and Papyrus were used in the case studies.

From the case studies (Liebel et al. 2016, 2017) we find that students can use industrial modelling tools successfully, but require substantial coaching in how to use the tools, with a dedicated tool champion present in the course. We further find that the tools' inability to provide adequate feedback impacts the tool acceptance. In an attempt to connect industry practice to education, Whittle and Hutchinson (2011) present essential differences between industry practice and education concerning software modelling. The authors find that modelling education is more UML-centric. In contrast, the industry is placing more emphasis on abstract models, that is, working at the meta modelling level with in-house domain-specific languages and code generators. Furthermore, in industry, it is more common to use a bottom-up approach to adoption, whereas modelling is typically taught top-down. In the bottom-up approach, developers try different modelling aspects to refactor existing modules and not think about the whole system abstraction unlike the top-down approach, where the typical development starts with modelling the system requirements.

3 Research Method

Our goal is to investigate students' perceptions, especially their challenges in learning and applying software modelling in a university environment. To do so, we conducted a multiple-case study. Runeson et al. (2010) define a case study as an "empirical investigation of a software engineering phenomenon within its real-life context" where evidence can be drawn from multiple sources, more importantly from real-life experiments involving human participants (Wohlin 2021). The phenomenon we study is the experience of students learning software modelling at university. In total, we interviewed 16 students from 3 different universities and 5 different courses. Additionally, we interviewed instructors of the 5 courses we considered. Three courses were at bachelor level and two were from masters level. In addition to the interviews, we consulted assignment details for the courses.

Whether students can or should be used as study participants is a long-debated question in SE (Falessi et al. 2018; Runeson 2003; Salman 2015; Sjöberg et al. 2002). In particular, students can be valid study participants under certain conditions, e.g., if novice software engineers are studied (Falessi et al. 2018). In our case, we aim to study how students perceive modelling, in contrast to existing studies that investigate industrial cases. Therefore, the choice of students as study participants is valid.

The study setup is described in detail in the following sub-sections.

3.1 Case Study Design

3.1.1 Pilot Study

Before starting the actual study, we conducted a pilot study with three interviewees, two doctoral students and one bachelor student. Of the two doctoral students, one has prior knowledge and experience of software modelling. The other student has no experience in software mod-

elling. We selected the combination because we wanted to check how our questions would be received by different persons with various software modelling background, as we are dealing with students from different countries, courses and backgrounds. Then we conducted another interview with a bachelor student who had taken a modelling course at our university. Our focus was to check the interview guide. Also, the pilot study helped us to estimate the overall interview time.

3.1.2 Cases and Recruitment

We designed a multiple *embedded* case study (Runeson et al. 2010) with five cases, each case is represented by a course-university pair. Table 1 shows the cases and the interviewees per case (using anonymous identifiers). Initially, we contacted the instructors of the courses based on our own contacts, and once they agreed, we advertised the study to students of that course. C1 was selected through convenience sampling, as it is a course at our home university. It is a typical "UML course" in the sense that basic UML diagrams are introduced. The course starts with user interface methods and prototype design; later on, students learn about software modelling, including UML use case, class, sequence, and activity diagrams. Assignments related to these diagrams are given. The course has two instructors, where I2 was responsible for the modelling part. However, both I1 and I2 were in charge of the exams. Therefore, we interviewed both instructors. Neither of the two instructors is active in modelling research. C2 was selected as a comparative case, as it was another typical "UML course" taught by the same instructor I2 as C1, at the same university. Compared to C1, C2 concentrates mainly on software modelling, focusing more on UML in the assignments. In C2, students start with user requirements, but unlike in C1, there is no interface or prototype design. Instead, students focus on use case diagrams, followed by class, state, activity and component diagrams. In C2, students get more time and opportunities to learn the UML diagrams. Hence, the quantity and quality of the assignments and the overall difficulty level are higher than C1. C1 and C2 are both bachelor courses.

However, C2 is located in the software engineering program and C1 in computer science. Our selection of courses is based on mainly two properties: courses provide UML lessons (from the basic level, like C1, to detailed, like C2), and courses have opportunities for the students to perform modelling assignments. After C1 and C2, C3 to C5 were selected as revelatory cases, as they have those two properties, but additionally, they exhibit one or several

Table 1 Case summary

Code	Level	Course Description	Students	Instructors
C1	Bachelor	Software analysis and prototype design using user interviews	C1_(S1, S2, S3, S4, S5)	I1, I2
C2	Bachelor	Software requirement analysis and design	C2_(S1, S2, S3)	I2
C3	Bachelor	Software requirement specification and design	C3_(S1, S2)	I3
C4	Master	System specification, design and testing	C4_(S1, S2, S3)	I4
C5	Master	Software architecture design and quality analysis	C5_(S1, S2, S3)	I5

differences from C1 and C2. One of the main differences is the change of university/country. We considered change of university/country as the different curricula and styles of teaching add variability to the case study. C2 and C3 are similar in terms of the course syllabus. Both courses teach system design and UML from scratch. C4 and C5 are courses in the same university and are master courses. While, in C4, UML is taught from scratch, C5 focuses on using UML diagrams for architecture and testing.

Apart from C1, all four courses include implementation tasks. All five courses use service-oriented domain examples like bike renting systems, dating websites, and hospital management systems for assignments. We did not explicitly ask about the programming language used for implementation since it was out of our research scope. Regarding tooling,

Table 2 Detail case description

Cases	Level	Syllabus	Assignments	Modelling Tools
C1	Bachelor	UML is taught from scratch. "It is an introductory course to modelling, focusing on basically learning how to use the UML standard"	Assignments include drawing basic UML diagrams. No programming	No specific tools, but mostly draw.io
C2	Bachelor	UML is taught from scratch. The course covered, "from the first specifications or initial ideas of a system and all the way up to a point where you're ready to start implementing based on the structure and the models that you have created of the systems"	Assignments include drawing a variety of UML diagrams for different problems, the number of assignments is higher than C1	No specific tools, but mostly draw.io
C3	Bachelor	UML is taught from scratch. The course includes," eliciting the requirements, making domain models and then move on into design models, and then we model behavior and do some sequence diagrams and state diagrams, sequence states. We go a little bit into component modeling as well"	Assignments include requirement elicitation using use case diagrams, domain modelling using class diagrams, system design using sequence/activity diagrams	No specific tools
C4	Master	The UML is taught from scratch. The course involves, "teaching UML modelling to use in advance topics like requirements engineering, system testing. We try to cover from the ground elicitation to design every detail"	Assignment includes designing the system using UML models, implement them and providing project demos to describe the relation between design class and its implementation	No specific tools
C5	Master	UML is not taught from scratch. "It's an introductory course on architecture, I tell them a lot about that...and then expect them to use UML for it. In this course, basically, they come pretrained in what in a previous course they have been introduced (to UML)"	Assignments include architecture design using UML modelling and implementation.	No specific tools

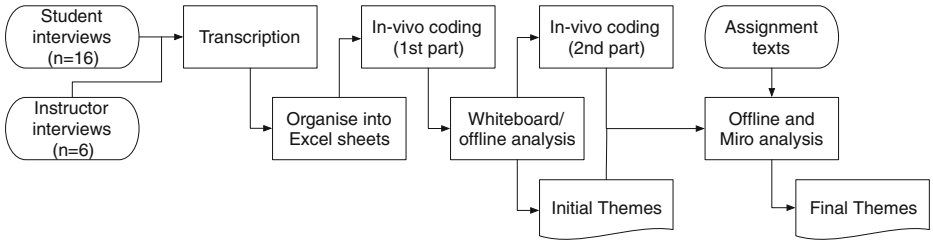


Fig. 1 Data Analysis Process. Rounded rectangles denote data sources, regular rectangles denote activities, rectangles with wavy bottom line denote artefacts, and arrows denote information flow

there are no specific instructions; students used tools like draw.io, PlantUML, Magicdraw as per convenience. All five courses are taught in English. The detailed description of all five courses is provided in Table 2. For describing the syllabus, we use each instructor's own statements. And for the assignments, we try to provide an overview to specify the depth of each course and how modelling is examined. Please note that, bachelor courses (case C1, C2 and C3) had written examinations as well.

Another difference between C1, C2 and the rest of the cases is the amount of industry experience among students. We interviewed a total of 16 students. Seven among them have industry experience, two bachelor students (C1_S3 and C3_S2) and five out of six master's students (C4_S1, C4_S2, C4_S3, C5_S1 and C5_S2). Four were employed as "software engineers" (one student was still working during the interview), one worked with databases, and two worked in a private organisation on various projects. We did not plan to get students with and without industry experience since participation was voluntary. And since the years and fields in the industry among students varied a lot, we did not take that as one of the primary attributes of analysis. We focused on students' overall experience in their respective courses. Despite their academic and technical background, we did not find significant differences between students' statements with industry backgrounds and those without.

Participation by the interviewees was voluntary and instructors were not aware of who participated in the interviews. We asked interested students to contact the researchers via mail, and once students initiated the contact, we sent them an interview meeting invitation and a consent form. All interviews were online. To appreciate students' participation, we offered them movie vouchers or did a donation to charity on their behalf.

3.2 Data Collection

Based on the pilot study, we decided to conduct interviews for approximately 40 minutes. The student interviews were divided into two set of questions: **introductory questions** and **model questions**. In the **introductory questions**, we added questions about a student's background, previous work experience with software models. The **model questions** includes students' personal experiences during courses, challenges in learning modelling, completing assignments and overall perception of software modelling. The instructor questionnaire includes details about the course structure, assignments, feedback from students and the instructor's challenges with the course. All interviews were conducted remotely through Zoom or Microsoft Teams, and their built-in recording features. Before recording, we asked permission from each interviewee and received their signed consent form. Interviewees could additionally give their consent that we would publish their anonymised interview transcript. 13 interviewees consented to publishing. The interview guides and the transcripts can be found on Zenodo Chakraborty and Liebel (2022).

3.3 Data Analysis

The data analysis process is depicted in Fig. 1. After conducting the interviews (steps 1a and 1b in Fig. 1), we transcribed the interviews using transcription services.[1] In case of automated transcriptions, we post-processed the transcripts to improve their quality. The two researchers then applied in-vivo coding separately on each interviewee transcript. In-vivo coding is one of the first cycle coding used for qualitative analysis (Saldaña 2015), where literal spoken words are used as codes, instead of assigning own terms (open coding). This helps reduce the risk that interviewees' words are misrepresented at the coding stage. In our case, interviewees were from different countries and cultural backgrounds, and thus used varying vocabulary/terminology to describe their personal experiences. Therefore, in-vivo coding was suitable to represent their personal experience. Finally, in-vivo coding is a suitable coding approach for early researchers (Saldaña 2015), as the first author.

Both researchers coded the interviews individually and discussed the outcomes for the first five interviews (C1_S1, C1_S2, C1_S3, C1_S4 and C2_S1). After this initial stage, we jointly grouped the codes into initially 10 themes. These themes related to benefits and challenges of modelling, as well as general categories referring to course feedback and concerns. Next, we analysed the remaining interviews from C2 and two from C3. After that, we proceeded with C4 and C5, our two cases with master's students. We refined the themes as the analysis progressed and more interviews were added. While working on the student interviews from C4 and C5, no substantially new themes emerged compared to the initial analysis. Therefore, we decided to stop at this point with a total of 16 student interviews. Once we finalised the themes based on student data, we coded the instructor data and compared it with the existing themes. We used the instructor interviewees for three purposes, namely (a) to understand the course context, (b) to better understand the student perceptions and to check whether the instructors had the same views and (c) to obtain ideas for potential best practices. For sorting quotes and themes, we used the online whiteboard tool Miro.[2]

As a second data source for our analysis, we used course assignments to cross-check themes and quotes that related to, e.g., assignments, work load, or grading. We contacted the instructors after their respective interviews for assignment details, which included the assignment structure, time, instructions, and point distribution. Only three instructors (C1/I1, C3/I3 and C4/I4) responded. However, the majority of students in our sample are in those courses. For the other two courses, we used the instructor interview data, and tried to gather as much information about course timing, assignments, and grading as possible (see questions 3 and 4, in Appendix A2).

We checked the assignments for the problem motivation, the actual tasks, the required problem domain knowledge, clarity of instructions and the total time dedicated to modelling tasks. We then compared these aspects to statements made during the interviews.

The total time of data collection and analysis was approximately one year. Hereby, the first five interviews took the majority of time, as they included establishing joint coding practices and discussing initial themes.

In the next section, we discuss the results and answer the RQs.

[1] We used Konch (https://www.konch.ai/) and Go Transcript (https://gotranscript.com/).

[2] https://miro.com/

4 Results

In this section we report and discuss the findings of our case study. The resulting themes related to modelling benefits are listed in Table 3, with example quote for each theme. Similarly, Table 4 lists the themes related to modelling challenges.

4.1 Student Perception of Software Modelling (RQ1)

Based on the interview data, we observe that students see specific benefits of modelling, primarily for areas where informal models might be sufficient, such as obtaining a system overview or conceptual understanding of the problem domain. However, many are skeptical as to what value modelling has for detailed system design. In the following, we discuss the students' perceptions and provide adequate interview quotes that support the categories. An overview of how many students mentioned the perceived benefits is depicted in Fig. 2.

4.1.1 Being on the Same Page

Several students find that modelling helps their groups communicate better with each other. Students mention modelling helps them to express their ideas, share work and make decisions as a group. For example, students stated:

> *"I think it's important for documentation, as well as getting everyone on the same page"* — C2_S1

> *"It's a lot easier to work together when you have the diagrams."* — C1_S5

Table 3 Final themes related to modelling benefits, with a short description and a sample quote associated with each theme

Theme	Description	Example statement
B: Same Page	Benefits of modelling in communication and coordination within teams.	*I think it's important for documentation, as well as getting everyone on the same page*
B: Better Planning	Modelling helps in planning ahead, for understanding the requirements, potential design alternatives, but also to plan implementation.	*But the thing I liked this is when you feel like you have a plan and you know what are you gonna do*
B: Better Understanding of Code	Benefits of modelling to help understanding a code base without reading the code in detail.	*I really like the idea of seeing it as a language to talk about code, because I will never read someone's code if they ask for feedback-*
B: Maintenance/ Documentation	Benefits of modelling as a way to help maintaining a system and to document what it does.	*Usually, it's other people who will maintain software, so they need to read what you make if they need to understand.*
B: Personal Preferences	Benefits related to individual preferences.	*Yeah, yeah we used that because it was um..prettier, than this is the whole perfectionism thing with um, it was prettier than coding*
B: Doubtful Benefits	Doubts regarding any benefits of modelling.	*[..]]you can't be mastering everything and [I am] more into implementation rather than modelling things*

Table 4 Final themes related to modelling challenges, with a short description and a sample quote associated with each theme

Theme	Description	Example statement
C: Unclear Expectations	Confusion on what the purpose of modelling is, often as expectations are not clearly communicated.	*At the end it's like 'Oops, maybe I should have designed', but then I wouldn't know how to design it.*
C: Irregular and Unclear Feedback	Feedback is provided too seldom, is not clear, or is restricted to formalities, such as the diagram syntax.	*Writing for hours and then a teacher be like no this is wrong*
C: Lack of Expertise in the Problem Domain	Students are provided with assignments in a domain they are unfamiliar with. Therefore, they struggle to relate their learning to something known.	*we didn't have much experience in programming and designing and modeling the app, I think the biggest challenge was in the first week that we were just flowing with our ideas how the app should look like without actually knowing how the end picture of the class diagrams should look like*
C: Time and Repetition	Modelling consumes time, and requires repetition to master. These two factors are often in conflict in university courses, and students struggle to see the value of repeatedly improving their models..	*Later on, we had to fix it at least 10 times. After writing the app, we had to go back to our diagrams and redo them how our final vision was*
C: Notation	Struggles with the complexity of the UML notation.	*One thing that comes up is with the state diagram and activity diagram, which one is supposed to do what*
C: Tooling	Modelling tools can cause numerous difficulties, such as poor usability.	*I hate PlantUML. I can't read the diagram that comes out of it. It's all over the place*
C: Lack of Cooperation	Challenges due to difficulties in collaborating in teams, and due to lack of professionalism in students' attitudes.	*[..] one or two would make the model and the rest maybe don't understand. Somehow, I don't know how to solve this problem.*

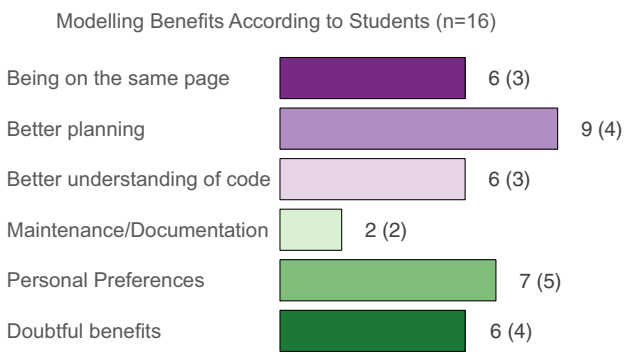


Fig. 2 Benefits as supported by the student interviewees. Numbers over the bars represent the total amount of students mentioning the benefit, and the number of cases in which students mentioned the benefit in parenthesis

"It helps every group member to know what exactly they want" — C3_S2

4.1.2 Better Planning

Students find modelling helpful in planning the development, i.e., designing systems. Students appreciate that they can visually map the system and plan the development through modelling.

"But the thing I liked this is when you feel like you have a plan and you know what are you gonna do" — C1_S2

"Class diagrams are really, really helpful. Before starting your project...because you can see and have the view of your classes" — C5_S3

4.1.3 Better Understanding of Code

In relation to programming, some students found models helpful to get an overview of the code on a higher level of abstraction. That is, they stated:

"I really like the idea of seeing it as a language to talk about code, because I will never read someone's code if they ask for feedback" — C3_S1

"If you have a diagram for it, we would get a better understanding of the code itself." — C2_S3

An aspect of this is a top-down modelling approach, where a better understanding of the system is obtained through breaking down the system in steps, as noted by some of our interviewees.

"If you are making a program that has more than 200 lines of code then you probably gonna have to model it." — C1_S1

"that was a very nice experience of seeing how you start with a kind of vague idea and then start to break it down more concrete classes" — C3_S1

4.1.4 Maintenance/Documentation

Students find modelling to be helpful to prepare for future tasks, such as maintaining and documenting the product. However, in our interview data only students with industry background expressed this benefit.

"Usually, it's other people who will maintain software, so they need to read what you make if they need to understand." — C1_S3

"Unless you don't have the documentation, you will end up with just a total mess" — C4_S2

While looking at the assignment details and interview data regarding that, we realized none of the courses explicitly mention maintenance/documentation. The assignments ask for an overview of the diagrams, but there aren't any requirements for documentation. The importance of maintaining a document or how to do that still needs to be clarified for students.

4.1.5 Personal Preferences

In addition to the benefits stated above, some students use models in terms of personal preference. For example:

> *"Yeah, yeah we used that because it was um..prettier, than this is the whole perfectionism thing with um, it was prettier than coding"* — C1_S2

> *"It helps me basically get my idea out there a lot better. Basically, when I'm starting a project, I like modeling the higher level and just going deeper and deeper and deeper."* — C2_S1

4.1.6 Doubtful Benefits

Despite experiencing benefits, several students are doubtful whether the benefits of modelling outweigh the issues, and if they will apply models in the future. We received several statements of students saying they will most likely not apply models in the future.

> *"I think it's a great experience that we can have this now. Not in the future, in our jobs"* — C3_S2

> *"[..]]you can't be mastering everything and [I am] more into implementation rather than modelling things"* — C4_S2

Interestingly, both statements above were made by students with industrial software development experience. The same students however admitted that in their experience they were tasked with implementation only, and not with high-level tasks such as system design or requirements engineering.

One of the instructors confirmed that many of their students would question the application of modelling in industry:

> *"It's the same experience that I have when people talk about this course, they're asking, 'Is anyone using it?"'* — I2

We further discuss the challenges with modelling in the next section.

4.2 Modelling Challenges (RQ2)

Despite benefits observed by the majority of the students, they experience several challenges related to modelling. These relate, among others, to tooling, how to choose the right notation, what to express in the models, and how to apply it to unfamiliar domains. We will discuss these challenges in the following, focusing on the student lens, and complementing their views with the instructor perspective. Overall, we extracted 8 types of challenges. The support by the different interviewees is depicted in Figs. 3 and 4.

4.2.1 Unclear Expectations

In university modelling courses, students typically receive assignments to create models. However, they often struggle to understand what is expected of them, and how to create the models.

> *"We started coding and at the end of the day, we didn't know what we did"* — C4_S3

In particular, this is caused by a lack of knowledge in programming, software architecture, and other practical skills necessary to envision a "good" design:

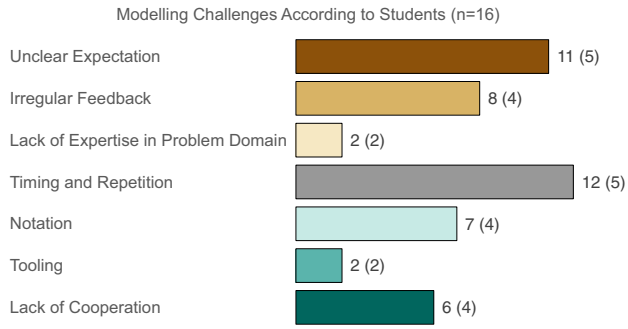


Fig. 3 Student challenges as supported by the interviewees. Numbers over the bars represent the total number of students mentioning the challenge, and the number of cases in which students mentioned the challenge in parenthesis

> *"At the end it's like 'Oops, maybe I should have designed', but then I wouldn't know how to design it. "* — C3_S1

Additionally, assignments are often formulated in a way that is in contrast to how they are assessed, e.g., by asking students to create a *prescriptive* model, which is then assessed by how closely it resembles the final code, i.e., in a *descriptive* way. Students quickly pick up this discrepancy in grading and optimise their efforts towards the grading. That is, they take shortcuts initially, and simply update the model later to reflect their actual code.

> *"we had to, we had to basically just create a class diagram out of the code that they wrote, instead of writing a code out of the class diagram"* — C1_S1

4.2.2 Irregular and Unclear Feedback

Students are dissatisfied with the type of feedback they receive. The evaluation criteria often demand *"Diagrams are well structured and provide good overview"*- assignment details from C3, *"Explanation of good design suggestions"*-assignment details from C1, which is

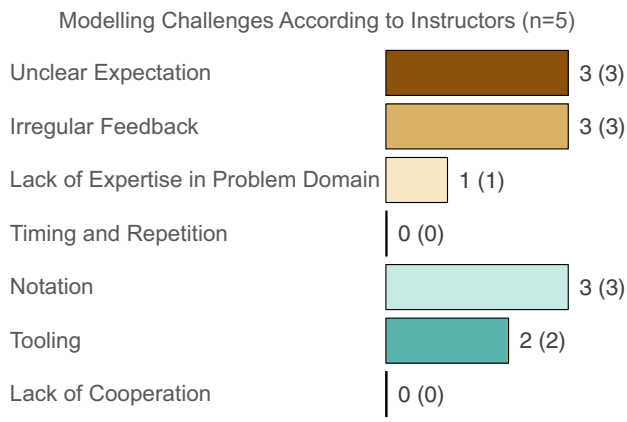


Fig. 4 Student challenges as supported by the instructor interviewees. Numbers over the bars represent the total amount of instructors mentioning the challenge, and the number of cases in which instructors mentioned the challenge in parenthesis

vague and do not leave space for constructive, detailed feedbacks. As modelling is often a qualitative task, teachers often revert to giving feedback on objective things such as the diagram syntax. However, this is not perceived as useful feedback.

> *"So no one can say anything to us. It's good or not. Just we should follow some rules about the diagram, for example, what is solid line, what is dashed line."* — C5_S1

Instead, students would like more "holding hands" with regular feedback.

> *"Maybe because it's the first year, this is the first introduction to the subject, so people miss a lot of points, but if it had another level, teach people more, and take their hand more to do these diagrams themselves, that would be good"* — C1_S3

In particular, since there is no automated feedback as, e.g., in programming, students require timely and regular feedback.

> *"writing for hours and then a teacher be like no this is wrong"* — C1_S1

> *"Reply to emails more than five or six hours"* — C4_S1

The regularity and quality of feedback was further complicated by Covid-19, as all five cases used remote teaching.

4.2.3 Lack of Expertise in the Problem Domain

Deep knowledge of the problem domain is necessary to perform many software engineering tasks in a satisfactory manner, e.g., programming (Oliveira et al. 2000) or program understanding (Rugaber 2000). If this domain knowledge is lacking or entirely missing, software engineering tasks cannot be performed well. Allowing students to work with a known problem increases their interest and participation. Several students stated that, in addition to just learning modelling, they were also lacking domain knowledge and programming experience. Therefore, they were unable to relate their models to familiar concepts.

> *"we didn't have much experience in programming and designing and modeling the app, I think the biggest challenge was in the first week that we were just flowing with our ideas how the app should look like without actually knowing how the end picture of the class diagrams should look like"* — C1_S4

Compared to other introductory topics, such as programming, this makes modelling particularly complex: Students are expected to learn a new concept (modelling), neither having problem domain nor software design/architecture experience. In addition, the models used in the courses we studied do not provide any kind of automated feedback, such as compiler/runtime error messages in programming. This leads to feelings of "just drawing something" in students, without an anchor to connect their models to.

In a similar direction, the chosen problem domain might also affect how representative the learning is for actual systems in industry, or how suited modelling is for the given problem. For instance, one student noted:

> *"we got to know all the basics and the whole idea and how it's supposed to look like, but not how to use it in the future on the bigger picture and the scale of the app"* — C1_S4

4.2.4 Time and Repetition

Students complain that modelling takes time, and courses are tightly scheduled. Students feel pressure due to the bulk of assignments, especially when they are not fully aware of what is expected from them. Given that pressure, they try to prioritise and minimise effort where

possible. However, arguably, learning how to model will require the students to make many mistakes and correct them in iterations. Together with the lack of feedback mentioned above, students perceive these iterative improvements as a waste of time.

> *"You run it once and then probably change. It's just a waste"* — C3_S1

> *"we designed something, it didn't work, so we needed to change it "* — C2_S1

> *"Later on, we had to fix it at least 10 times. After writing the app, we had to go back to our diagrams and redo them how our final vision was"* — C1_S4

Interestingly, these students did not perceive that iterative changes helped them understand the problem domain better, but rather saw it as a burden with unclear benefit. One of the reasons is courses contain individual assignments without a continuation or connection between themselves for some cases (C2 and C3). For each problem students are instructed to draw a specific diagram and then move on to something else. An exception we observed for C4 and C5 since, in C5, students are applying what they learned in C4.

4.2.5 Notation

Siau and Loo (2006) reported different learning challenges with UML over a decade ago. In our cases, we observe similar challenges. Students find it hard to remember the syntax of different diagrams and their purpose, pointing to a lack of prior knowledge of UML.

> *"many different types of diagrams to represent the same thing"* — C1_S1

> *"The hardest part about drawing UML diagrams is always remembering what exactly the syntax is"* — C2_S1

Specifically, our interviewees are most often pointing out UML state machine and activity diagrams as confusing.

> *"I have this image in my head where the state diagram is not the UI actions. I can't wrap my head around it."* — C2_S3

> *"One thing that comes up is with the state diagram and activity diagram, which one is supposed to do what. "* — C1_S5

The instructors confirm this view.

> *"here's a bit of struggle in between the concepts that they express in a diagram and the particular diagram types"* — I3

> *"they've been mixing those pretty well up [state and activity diagrams]"* — I2

One instructor offered the explanation that this relates to a lack of experience with object-oriented programming in general.

> *"they don't have a lot of experience using object-oriented programming."* — I2

This quote reinforces our discussion above, that students are exposed to various unknown activities when learning how to model, and therefore lack knowledge they can anchor their modelling experiences to.

4.2.6 Tooling

None of the five cases we investigated had any requirements for using a particular modelling tool. Correspondingly, students did not voice any significant tool challenges, other than their dislike for the tooling interface. Students used multiple tools for their assignments, e.g., Draw.io, PlantUML, and Lucidchart. Also, some students used plain pen and paper for drawing and communicating their diagrams. Students appreciate this choice.

> *"I think actually giving people the freedom to do what they want with the tools is really nice, because some people actually liked drawing it on an iPad or something"* — C2_S1

In particular, some students also showed initiative at choosing different tools depending on the situation.

> *"When I did the prototype of my app, I used the Figma app on the internet. It's really good to visualize the prototype. Then we used the Lucidchart and draw.io, I think it's diagrams.net now. This is for the diagrams. Then we just used the whiteboard to draw it if we needed to because we just gathered together and we'd just throw it on the whiteboard and then we just translated it into diagrams or Lucidchart"* — C1_S4

While offering choice in tooling clearly addresses many tool issues, it has the limitation that it only works if the course and the course assignments do not rely on specific tool features, such as code generation or simulation capabilities. This is a limitation in our study, as all our cases introduced UML only as a means to document, plan, and to communicate, without any automated processing of models in the form of model transformation or other facilities.

4.2.7 Lack of Cooperation

In all five cases, students had to cooperate in teams. In fact, modelling was in all cases also intended as a way to facilitate communication in the group. However, in practice they struggled to cooperate and to appreciate the use of models for communication purposes.

> *"We are six, for example, but one or two would make the model and the rest maybe don't understand. Somehow, I don't know how to solve this problem."* — C1_S3

> *"Some people don't like to draw., but we're supposed to work together."* — C5_S1

4.2.8 Summary

Overall, we observe that students struggle with various aspects of models (as depicted in Figs. 3 and 4). Apart from classic issues such as learning an unknown notation (be it modelling or not), we find that students lack a connection to previous knowledge. That is, due to a lack of domain, design and programming knowledge, and a lack of automated feedback from modelling tools, they cannot create an anchor. Essentially, they produce a model that they cannot judge on any other level than notation. One student summarised this sentiment as follows:

> *"The diagram is supposed to speak for yourself [sic]. But we didn't understand what it was saying"* — RU_G_S1

5 Discussion

Based on our results, several recommendations for modelling education can be made. We discuss these in Section 5.1. While some of the recommendations relate to modelling, they can also be attributed to poor pedagogy, which we briefly discuss in Section 5.2. Finally, we summarise to what extent our findings in education compare to industrial practice in Section 5.3.

5.1 Recommendations for Modelling Education

We observed five benefits (Fig. 2) and seven challenges (Fig. 3) related to modelling. Among the seven challenges, the three most voiced challenges are *Unclear Expectations*, *Timing and Repetition*, and *Irregular Feedback*. We believe that **iterative, project-based learning (PBL)** (Krajcik and Blumenfeld 2006) has the biggest potential to address these challenges. An iterative way of working allows for regular feedback. Repetition can be achieved by **limiting the amount of diagram types used**, thus focusing on a few diagrams that are improved in iterations. **Refining and improving diagrams over several iterations** allows for expectation management, as the diagrams' purpose becomes clearer over time. Furthermore, it might reduce the assignment pressure, thus potentially addressing *lack of cooperation*, as student are less likely to optimise workload by distributing the assignments. Working in groups on a project also allows to maintain the benefits our participants see in models, i.e., using them to align their expectations of the system under development, to plan ahead, and to maintain and document the system under development.

Regarding assessment, it is essential that **stated model purposes and grading rubrics are aligned**. In several cases, we observed that modelling assignments were formulated as planning tasks, but assessed later on based on how well a model described the resulting system. This led students to simply update the models prior to submission, with the intended purpose ultimately lost. We believe this also contributes strongly to the *Unclear Expectations* challenge we observe.

A way to address a lack of feedback from modelling is through automated means, e.g., by using code generation. However, depending on where a course is placed in the curriculum, students might not yet appreciate the value of code generation or other advanced topics, such as simulation or general model-driven engineering concepts. Instead, they could perceive it as a form of restriction, as the joy of programming is taken away from them. Additionally, we observe that one of the benefits students regularly express are various *personal preferences*, i.e., they like modelling as it allows for freedom in expressing themselves using a graphical notation. We believe this is a further argument why **using code generation should be discouraged in early modelling courses**. Instead, modelling tasks that allow for more freedom in notation and tooling can help students to value modelling, which could encourage them to pick up more advanced techniques later on.

A regular-voiced challenge in our data is *lack of expertise in the problem domain*. To address this challenge, educators need to **use familiar problem domains**, potentially chosen by the students themselves. To some extent, student choice could also be limited, e.g., by approving chosen problem domains or providing a list of characteristics every domain needs to fulfill. This would allow to avoid typical pitfalls, such as overly simplistic domains or those that are not particularly suitable for the intended tasks. Paige et al. (2014) argues for using "fun" domains. While this does not contradict our recommendation, we would caution to use

those to which students can relate and have some familiarity at least. Doing so would allow them to incrementally learn more about the domain, while not being entirely lost initially.

Modelling tools have been reported consistently as a concern in studies on software modelling, both related to industry and education. In contrast, only one of our participants stated an issue with a modelling tool. While, to some extent, the absence of reported issues with tooling is likely due to the courses' focus on "drawing" diagrams, and the free tool choice, we do believe that this freedom has its advantages, as the chance is higher that students experience a first positive modelling experience, rather than being frustrated by tooling issues. Therefore, in line with the recommendation to refrain from using code generation, we suggest to **allow for flexible modelling tool choice**. None of the cases used tools that are specifically designed for education, such as the Umple modelling tool (Garzón et al. 2015). Since surveys with students have shown positive results (Lethbridge et al. 2011), we believe studying such a course in depth using qualitative methods could yield further insights that are complementary to ours. However, we also believe that the use of software modelling tools tailored towards education is currently more the exception rather than the rule.

Several of our findings that relate both to perceived benefits and challenges of models are specific to the selection of courses we studied. That is, in all five courses, models are created for planning, design, and communication purposes. In contrast, topics that require formal models, e.g., for code generation, formal verification, or simulation, are not included. This certainly limits how general our findings are with respect to software modelling as a whole. Nevertheless, we believe that a large percentage of computer science students worldwide get exposed to software modelling through similar courses. As such, our findings have the potential to be transferable to many students. Similarly, even if students take more advanced courses on software modelling, their beliefs and preconceptions about modelling can be irreversibly shaped by their first exposure to the topic.

5.2 Pedagogical Influence

Several of the challenges perceived by students might not be caused by software modelling as a course topic, but rather by the pedagogical quality of the courses. We did not try to assess the quality of the individual courses, and given the chosen research method we cannot control for it, either. However, we cover a variety of expertise in modelling on the lecturers' side, ranging from instructors that do not work on models in research and do not have a dedicated interest in modelling, to instructors that publish in software modelling venues. Therefore, we do not believe that the students' experiences can be related to issues in teaching only. The challenges students expressed are likely triggered by a combination of individual teaching styles, the course syllabus and the placement of the course in the overall program, the types of assignments and their assessment, and student preferences. As such, we believe it is rarely possible to claim that challenges arise due to poor teaching.

5.3 Education and Industrial Practice

While this study focused purely on the educational context, it is interesting to discuss potential similarities and differences to industrial practice. First, we observe that models are appreciated for communication purposes and to handle system complexity in our cases, something that is also reported in industry (Gorschek et al. 2014; Liebel et al. 2018a; Störrle 2017). Other benefits reported in industry, such as simulation or verification capabilities do not apply in our cases, since all five courses used modelling only on an informal level to express models

for planning, communication, and coordination. Similarly, it is hard to reason about improvements reported from industry, e.g., in terms of productivity (Baker et al. 2005; Mohagheghi and Dehlen 2008).

Tooling issues are a frequent topic in industry, e.g., reported in Hutchinson et al. (2011a, b); Liebel et al. (2018a, b); Whittle et al. (2013, 2014). In contrast, only one of our interviewees raised tool issues. One explanation for this observation is clearly that none of our courses mandated a dedicated modelling tool for modelling, but instead left that choice to the students. Similarly, while models were often assessed for semantic correctness, the specified purposes of the models did not require syntactically or semantically correct models. Together with a lack of requirements for interoperability between tools, this removes most of the issues contemporary modelling tools have. Nevertheless, it is positive to observe that our interviewees did not finish their courses on modelling with the perception that modelling tools are bad, as is commonly reported in literature on modelling education, e.g., in Akayama et al. (2013); Liebel et al. (2016). Such a negative perception could lead to a reduced uptake of modelling in industry.

Our interviewees raise several challenges that relate to a lack of guidance, feedback, and clarity when it comes to modelling. While they primarily perceive this as an issue in how assignments are set up and how the courses are designed, the challenges resonate with those in industry. For instance, a lack of training and guidance is raised in several empirical studies on modelling in industry, e.g., Gorschek et al. (2014); Liebel et al. (2018a); Torchiano et al. (2013). Whittle et al. (2013) highlight that organisational and process factors play a major role in the use of MDE. Similar issues are raised by our interviewees in the educational context, namely that modelling assignments need suitable processes that allow for iterations and quick feedback loops. This is especially important as models were not used for automated tasks in any of our cases, i.e., there was no automatic feedback generated.

An important difference of the educational setting to industry is that students often lack both the technical background, e.g., in terms of programming experience, and the domain knowledge required for the example domains. Therefore, they struggle to contextualise the value and the quality of their models, often leading to a perception of doing a meaningless drawing task. Publications on modelling education sometimes argue for the use of realistic examples and caution against toy examples, e.g., Kolovos and Cabot (2016). However, also realistic examples have their pitfalls. While unrealistic domains are just that, unrealistic, they can also expose students to a known domain, or at least a low level of complexity due to a domain that is artificial and potentially more controlled and restricted. From this point of view, toy examples can be a suitable tool for modelling education. Given our findings in this study, we advise to use primarily example domains the students have sufficient knowledge of, as discussed above.

Finally, we observe that several benefits and challenges voiced by our interviewees are directly reflected in industrial surveys on the use of models, e.g., the usefulness of models for communication (Gorschek et al. 2014; Liebel et al. 2018a; Torchiano et al. 2013), or the sentiment that models are not worth the effort (Gorschek et al. 2014; Torchiano et al. 2013). While we are not able to answer this based on our study, we would like to highlight the possibility that these opinions are shaped during university education and then maintained later on. That is, **addressing these issues while educating modellers** could lead to a higher uptake and appreciation of modelling in industry. Similarly, **making the use of models in industry more transparent** could be beneficial, also for purposes where formal models might not be required, such as documentation or communication.

5.4 Validity Threats

We conducted an exploratory (Runeson et al. 2010) case study, where we primarily collected data through interviews. For the analysis, we leaned towards interpretivism. An interpretivist approach means that *"humans construct knowledge as they interpret their experiences of and in the world; rejecting the objectivist notion that knowledge is simply there to be identified and collected"* (Hiller 2016; Pascale 2010). A validity threat of our study is that the student's perception of modelling can be naive. Our interviewees are mostly beginners in modelling (bachelor students), and only a few have industry experience. Also, the knowledge among them varies a lot as some had very different types and amounts of industry experience, along with the different cultures the students belong to. Therefore, adopting interpretivism is suitable as we seek answers for our RQs through the interviewees' perspective, based on their knowledge of the addressed subject and cultural background. The understanding of our knowledge is, therefore relative to the person and their personal experience.

Following the work of Petersen and Gencel (2013), we present the validity threats of our work and the measures we have taken to mitigate them in the following.

5.4.1 Transferability

Transferability describes to what extent results from the study can be transferred to cases that resemble the case under study (Petersen and Gencel 2013). Cultural differences, the influence of teachers and their teaching practices, and the course subjects limit transferability in our study. We aim to ensure a substantial level of transferability by basing our analysis on five cases. Nevertheless, all five cases teach modelling for analysis tasks on a high level of abstraction, i.e., for requirements, architecture and design purposes. Specifically, none of the courses covers model transformation or other tasks that require formal models. Furthermore, all five courses are located in Northern European Universities, thus potentially limiting the applicability to other countries. Finally, perceptions of modelling are tightly connected to perceptions of the course. Specifically, the quality of teaching and the instructor's expertise on the topic could positively or negatively affect the studied perceptions. To avoid this, we selected both cases where the instructors do and do not have a research background in software modelling. Pedagogical quality is harder to assess. However, we did not get the impression from students that individual courses had a low level of pedagogical quality. Nevertheless, this might be a threat to transferability.

To further allow for transferability, we conducted a substantial number of interviews (21). We reached a saturation point in our themes after the 16 student interviews, and hence stopped including further cases or interviewees. That is, in the last batch of interviews (n=6) we analysed, no new themes emerged that we had not yet included in our analysis.

5.4.2 Credibility

Credibility describes to what extent findings have been distorted by the researchers (Petersen and Gencel 2013).

We tried to avoid distortion of the reported findings by grounding the analysis in in-vivo codes directly taken from the verbatim interview transcripts. Furthermore, we performed member checking to get feedback on the extracted themes from our interviewees. Only few interviewees answered this call, but those confirmed the credibility of the found themes. All interviews were recorded, and data analysis performed on the verbatim transcripts.

Additionally, we publish the anonymised transcripts of the 13 interviewees who consented to this. This should ensure credibility of the findings.

5.4.3 Confirmability

Confirmability describes the extent to which conclusions made by researchers follow from the observed data (Petersen and Gencel 2013).

To allow for confirmability, we presented the way of coding in depth. Furthermore, we give example quotes for each theme, and publish anonymised student transcripts. Note that this does not necessarily ensure reliability, i.e., that analyses conducted by other researchers would yield precisely the same results. We discussed themes and codes and aimed to find consensus in our analyses. Nevertheless, we allowed for some disagreements to account for subjective impressions or opinions, as is common in interpretivist research.

6 Conclusion

We conducted a multiple case study investigating the perceptions students have of software modelling. To do so, we conducted 21 interviews with students and instructors in five courses at 3 universities, and consulted assignment descriptions of three of the five courses. The results offer rich insights into how students perceive modelling, and based on that, we discuss several recommendations for modelling education. While related work on the topic exists, it is usually in the form of survey studies, opinion and experience papers, and industrial case studies.

Several findings from related work are confirmed in our study, e.g., that while seeing the benefit of modelling for planning or communication, many students perceive modelling as not worth the effort or as an outright waste of time. They struggle with the fast pace of courses and a lack of repetition and in-depth feedback that goes beyond the diagram syntax. However, we additionally find differences to existing literature, or deeper explanations of phenomena observed in related work. For instance, our interviewees reported almost no tooling-related challenges, potentially as they were allowed to choose their tool of choice in all cases, and as none of the five courses used any model-based techniques that would require a specific modelling tool. Finally, we find that students struggle to understand modelling, as they are often at the same time lacking knowledge in areas related to the course projects, i.e., the problem domain, the intended task (such as design or architecture), object orientation, and programming skills. As there is often no automatic feedback from modelling tools, this means students cannot anchor their models to any known domain.

Our study is of interest to university researchers, educators and professionals. We recommend iterative and project-based learning for modelling education so challenges like irregular feedback and unclear expectations can be minimised. Working with a project-based approach within a group will make students clearer about the purpose of the assignments and the diagrams used. Students will be able to embrace benefits like planning and maintaining better communication. We suggest limiting the number of diagram types used in a course, so students can make mistakes and not get overwhelmed by the pressure of assignments. By using familiar problem domains and giving students the freedom in notation and tooling, educators can help students understand modelling in their time and pace. Our study confirms many existing beliefs but also highlights new details that can be considered when teaching modelling, i.e., a more nuanced view on how the choice of a problem domain affects the students' perceptions of modelling. For modelling researchers, it can open up new directions on how

to improve guidance and training in modelling, something that also professionals regularly report as challenging. Finally, professionals can use our findings to better understand what experiences university graduates have with modelling, which benefits they perceive, and why this is the case.

Appendix

A Interview Guides

We present two interview guides in this section used for students and instructors. We scheduled 40 minutes for student interviews and 30 minutes for instructor interviews. Interviews were recorded with permission from the interviewees.

A.1 Student Interview Guide

The interview starts with a small introduction of 5 minutes presented by the interviewer which consists the following points:

- Asking permission to record the interview.
- Introduction of the interviewer.
- Explain the purpose of the study, the research questions.
- Explain the rules of the interview.

Following the introduction, the interview starts. It has two parts, first the *Introductory Questions*. The duration of this part is 10 minutes.

1. I1: Which degree are you currently pursuing? (in which major?)
2. I2: Have you worked in industry? If yes, what role/domain?
3. I3: Do you have any experience with modelling? What kind of experience do you have with modelling?
4. I4: Go in detail with the experience, ask about syntax, diagrams that they had used before.

In the second part, *Modelling Questions* we go in detail with the modelling challenges and the duration is a maximum of 25 minutes. Before going into the modelling questions, the interviewer explains software modelling and her research interest in software modelling. The reason is to make sure that the interviewee has a clear idea and can give concrete answers.

1. M1: What experience do you have in software modelling?
2. M2: What did you like from your experience?
3. M3: What challenges did you face during your experience?
4. M4: In your opinion what are the reasons behind these challenges?
5. (If needed): When you are using a modelling notation what typically troubles you?
6. M5: How was your experience with modelling tools? Tell me about the tools you have used in the past.
7. M6: What did you like and dislike about each of those tools?
8. M7: For which purposes/in which situations (in SE) do you think modelling is useful? When is this not the case?
9. M8: In your case can you tell me of an experience where you found modelling useful?
10. M9: What advantages and disadvantages do you see in modelling?
11. M10: Did you get enough knowledge about modelling from the courses you have taken on the topic?

12. M11: What was missing from those courses?
13. M12: How would the perfect modelling course look like?
14. M13: Do you have any comments or suggestions?

A.2 Instructor Interview Guide

1. Could you shortly describe which topics you cover in your course?
2. Regarding modelling, what are you covering in the course?
   (a) Are you teaching modelling notation (e.g., UML diagrams)?
   (b) Are you teaching how to draw those models (example: identify nouns/verbs in a text to draw class diagrams)?
   (c) Are you teaching how the models relate to code?
   (d) Are you teaching semantics of modelling notations?
3. Roughly how much time is spent on modelling in your course?
4. Regarding modelling, what is covered in assignments or exam?
5. How do you think students perceive the modelling part of the course?
6. What do you think are the things that students struggle with (with respect to modelling)?
7. What challenges are you facing when teaching modelling?
8. How do you think this could be improved?

Acknowledgements We would like to thank the interviewees for participating in the study.

Data Availability We publish 13 out of the 21 anonymised interview transcripts on Zenodo, https://doi.org/10.5281/zenodo.6913780. We did not receive consent to publish the remaining 8 transcripts.

Declarations

Competing Interests The authors declare that they have no conflict of interest.

References

Agner LTW, Soares IW, Stadzisz PC, Simão JM (2013) A brazilian survey on UML and model-driven practices for embedded software development. J Syst Softw 86(4):997–1005. SI : Software Engineering in Brazil: Retrospective and Prospective Views

Agner LT, Lethbridge TC, Soares IW (2019) Student experience with software modeling tools. Softw Syst Model 18(5):3025–3047. https://doi.org/10.1007/s10270-018-00709-6

Akayama S, Demuth B, Lethbridge TC, Scholz M, Stevens P, Stikkolorum DR (2013) Tool use in software modelling education. In: EduSymp@ MoDELS

Anda B, Hansen K, Gullesen I, Thorsen HK (2006) Experiences from introducing uml-based development in a large safety-critical project. Empir Softw Eng 11:555–581

Baker P, Loh S, Weil F (2005) Model-driven engineering in a large industrial context-motorola case study. In: International conference on model driven engineering languages and systems, Springer, pp 476–491

Bernonville S, Kolski C, Beuscart-Zephir MC (2005) Contribution and limits of uml models for task modelling in a complex organizational context: case study in the healthcare domain. In: Internet and information technology in modern organizations: challenges & answers, proceedings of The 5th international business information management association conference, IBIMA, pp 119–127

Burgueño L, Vallecillo A, Gogolla M (2018) Teaching uml and ocl models and their validation to software engineering students: an experience report. Comput Sci Educ 1–19 10.1080/08993408.2018.1462000

Chakraborty S, Liebel G (2022) Dataset: we do not understand what it says – studying student perceptions of software modelling. https://doi.org/10.5281/zenodo.6913780

Ciccozzi F, Taentzer G, Vallecillo A, Wimmer M, Famelis M, Lambers L, Mosser S, Paige R, Pierantonio A, Rensink A, Salay R (2018) How do we teach modelling and model-driven engineering? a survey. In: Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings, pp 122–129

Dobing B, Parsons J (2008) Dimensions of uml diagram use: a survey of practitioners. J Database Manag (JDM) 19(1):1–18

Falessi D, Juristo N, Wohlin C, Turhan B, Münch J, Jedlitschka A, Oivo M (2018) Empirical software engineering experts on the use of students and professionals in experiments. Empir Softw Eng 23. https://doi.org/10.1007/s10664-017-9523-3

Forward A, Badreddin O, Lethbridge TC (2010) 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)

Garzón MA, Aljamaan H, Lethbridge TC (2015) Umple: a framework for model driven development of object-oriented systems. In: 2015 ieee 22nd international conference on software analysis, evolution, and reengineering (saner), IEEE, pp 494–498

Gilson F (2018) Teaching software language engineering and usability through students peer reviews. In: Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings, pp 98–105

Gonnord L, Mosser S (2018) Practicing domain-specific languages: from code to models. In: Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems: companion proceedings, pp 106–113

Gorschek T, Tempero E, Angelis L (2014) On the use of software design models in software development practice: An empirical investigation. J Syst Softw 95:176–193

Hammouda I, Burden H, Heldal R, Chaudron MR (2014) Case tools versus pencil and paper. In: ACM/IEEE 17th Int. Conf. on model driven engineering languages and systems–educators symposium

Hiller J (2016) Epistemological foundations of objectivist and interpretivist research. Barcelona Publishers

Hutchinson J, Whittle J, Rouncefield M (2014) Model-driven engineering practices in industry: social, organizational and managerial factors that lead to success or failure. Sci Comput Program 89:144–161. Special issue on Success Stories in Model Driven Engineering. https://doi.org/10.1016/j.scico.2013.03.017

Hutchinson J, Rouncefield M, Whittle J (2011a) Model-driven engineering practices in industry. In: 2011 33rd International conference on software engineering (ICSE), pp 633–642. https://doi.org/10.1145/1985793.1985882

Hutchinson J, Whittle J, Rouncefield M, Kristoffersen S (2011b) Empirical assessment of mde in industry. In: 2011 33rd International Conference on Software Engineering (ICSE), pp 471–480. https://doi.org/10.1145/1985793.1985858

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

Kolovos DS, Cabot J (2016) Towards a corpus of use-cases for model-driven engineering courses. In: EduSymp/OSS4MDE@ MoDELS, pp 14–18

Kopach-Konrad R, Lawley M, Criswell M, Hasan I, Chakraborty S, Pekny J, Doebbeling BN (2007) Applying systems engineering principles in improving health care delivery. J Gen Intern Med 22(3):431–437

Krajcik JS, Blumenfeld PC (2006) Project-based learning. na

Lethbridge TC, Mussbacher G, Forward A, Badreddin O (2011) Teaching uml using umple: applying model-oriented programming in the classroom. In: 2011 24th IEEE-CS conference on software engineering education and training (CSEE&T), IEEE, pp 421–428

Liebel G, Marko N, Tichy M, Leitner A, Hansson J (2018a) Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. Softw Syst Model 17(1):91–113. https://doi.org/10.1007/s10270-016-0523-3

Liebel G, Tichy M, Knauss E (2018b) Use, potential, and showstoppers of models in automotive requirements engineering. Softw Syst Model. https://doi.org/10.1007/s10270-018-0683-4

Liebel G, Badreddin O, Heldal R (2017) Model driven software engineering in education: a multi-case study on perception of tools and uml. In: 2017 IEEE 30th Conference on software engineering education and training (CSEE T), pp 124–133. https://doi.org/10.1109/CSEET.2017.29

Liebel G, Heldal R, Steghöfer JP .: Impact of the use of industrial modelling tools on modelling education. In: 2016 IEEE 29th International conference on software engineering education and training (CSEET), pp 18–27. https://doi.org/10.1109/CSEET.2016.18

Mohagheghi P, Gilani W, Stefanescu A, Fernandez MA, Nordmoen B, Fritzsche M (2013) Where does model-driven engineering help? experiences from three industrial cases. Softw Syst Model 12(3):619–639

Mohagheghi P, Dehlen V (2008) Where is the proof? - a review of experiences from applying mde in industry. In: Schieferdecker I, Hartman A (eds) Model driven architecture - foundations and applications, lecture notes in computer science, Springer Berlin Heidelberg, vol 5095 pp 432–443

Moody DL (2010) The "physics" of notations: a scientific approach to designing visual notations in software engineering. In: 2010 ACM/IEEE 32nd International conference on software engineering, vol 2, pp 485–486. https://doi.org/10.1145/1810295.1810442

Oliveira K, Regina A, Rocha C, Travassos G, Menezes C (2000) Using domain-knowledge in software development environments. Tech. rep., CiteSeerX. https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.2007

Paige RF, Polack FA, Kolovos DS, Rose LM, Matragkas ND, Williams JR (2014) Bad modelling teaching practices. In: EduSymp@ MoDELS, pp. 1–12

Pascale CM (2010) Cartographies of knowledge: exploring qualitative epistemologies. Sage Publications

Petersen K, Gencel C (2013) Worldviews, research methods, and their relationship to validity in empirical software engineering research. In: 2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement, IEEE, pp 81–89

Raistrick C (2004) Applying mda and uml in the development of a healthcare system. In: International conference on the unified modeling language, Springer, pp 203–218

Reuter R, Stark T, Sedelmaier Y, Landes D, Mottok J, Wolff C (2020) Insights in students' problems during uml modeling. In: 2020 IEEE Global engineering education conference (EDUCON), pp 592–600. https://doi.org/10.1109/EDUCON45650.2020.9125110

Rugaber S (2000) The use of domain knowledge in program understanding. Ann Softw Eng 9(1):143–192

Runeson P (2003) Using students as experiment subjects - an analysis on graduate and freshmen student data.Proc 7th Int Conf Empir Assess Softw Eng

Runeson P, Höst M, Rainer A, Regnell B (2012) Case study research in software engineering – guidelines and examples. John Wiley & Sons Inc. https://doi.org/10.1002/9781118181034

Saldaña J (2015) The coding manual for qualitative researchers. SAGE Publications

Salman I, Mısırlı AT, Juristo N (2015) Are students representatives of professionals in software engineering experiments? In: 2015 IEEE/ACM 37th IEEE International conference on software engineering, vol 1, pp 666–676. https://doi.org/10.1109/ICSE.2015.82

Schmidt A, Kimmig D, Bittner K, Dickerhof M (2014) Teaching model-driven software development: revealing the" great miracle" of code generation to students. Proceedings of the sixteenth Australasian computing education conference-vol 148:97–104

Siau K, Loo PP (2006) Identifying difficulties in learning uml. Inf Syst Manag 23(3):43–51

Sjöberg DI, Anda B, Arisholm E, Dyba T, Jorgensen M, Karahasanovic A, Koren EF, Vokác M (2002) Conducting realistic experiments in software engineering. In: Proceedings international symposium on empirical software engineering, IEEE, pp 17–26

Stikkolorum DR, Ho-Quang T, Chaudron MR (2015) Revealing students' uml class diagram modelling strategies with webuml and logviz. In: 2015 41st Euromicro conference on software engineering and advanced applications, pp 275–279. https://doi.org/10.1109/SEAA.2015.77

Stol KJ, Fitzgerald B (2018) The abc of software engineering research. ACM Trans Softw Eng Methodol (TOSEM) 27(3):1–51

Störrle H (2017) How are conceptual models used in industrial software development? a descriptive survey. In: Proceedings of the 21st international conference on evaluation and assessment in software engineering, pp 160–169

Torchiano M, Tomassetti F, Ricca F, Tiso A, Reggio G (2013) Relevance, benefits, and problems of software modelling and model driven techniques-a survey in the italian industry. J Syst Softw 86(8):2110–2126

Walderhaug S, Stav E, Mikalsen M (2008) Experiences from model-driven development of homecare services: uml profiles and domain models. In: International conference on model driven engineering languages and systems, Springer, pp 199–212

Westphal B (2019) Teaching software modelling in an undergraduate introduction to software engineering. In: 2019 ACM/IEEE 22nd International conference on model driven engineering languages and systems companion (MODELS-C), pp 690–699. https://doi.org/10.1109/MODELS-C.2019.00105

Whittle J, Hutchinson J, Rouncefield M (2014) The state of practice in model-driven engineering. IEEE Softw 31(3):79–85. https://doi.org/10.1109/MS.2013.65

Whittle J, Hutchinson J (2011) Mismatches between industry practice and teaching of model-driven software development. In: International conference on model driven engineering languages and systems, Springer, pp 40–47

Whittle J, Hutchinson J, Rouncefield M, Burden H, Heldal R (2013) Industrial adoption of model-driven engineering: Are the tools really the problem? In: International conference on model driven engineering languages and systems, Springer, pp 1–17

Wohlin C (2021) Case study research in software engineering-it is a case, and it is a study, but is it a case study? Inf Softw Technol 133:106514. https://doi.org/10.1016/j.infsof.2021.106514

Chapter 8

Paper III: Exploring Actions, Interactions and Challenges in Software Modelling Tasks: An Empirical Investigation with Students

Shalini Chakraborty, Javier Troya, Lola Burgueño, Grischa Liebel
Under Review in Empirical Software Engineering (EMSE)
https://doi.org/10.48550/arXiv.2409.13656

Noname manuscript No.
(will be inserted by the editor)

Exploring Actions, Interactions and Challenges in Software Modelling Tasks: An Empirical Investigation with Students

Shalini Chakraborty · Javier Troya · Lola Burgueño · Grischa Liebel

Received: date / Accepted: date

Abstract Background: Software modelling is a creative yet challenging task. Modellers often find themselves lost in the process, from understanding the modelling problem to solving it with proper modelling strategies and modelling tools. Students learning modelling often get overwhelmed with the notations and tools. To teach students systematic modelling, we must investigate students' practical modelling knowledge and the challenges they face while modelling. **Aim:** We aim to explore students' modelling knowledge and modelling actions. Further, we want to investigate students' challenges while solving a modelling task on specific modelling tools. **Method:** We conducted an empirical study by observing 16 pairs of students from two universities and countries solving modelling tasks for one hour. **Results:** We find distinct patterns of modelling of class and sequence diagrams based on individual modelling styles,

S. Chakraborty
Reykjavik University
Menntavegur 1, 102 Reykjavík, Iceland
ORCID: 0000-0002-9466-3766
E-mail: shalini19@ru.is

J. Troya
ITIS Software, Universidad de Málaga
Blvd. Louis Pasteur, Málaga, 29071, Spain
ORCID: 0000-0002-1314-9694
E-mail: jtroya@uma.es

L. Burgueño
ITIS Software, Universidad de Málaga
Blvd. Louis Pasteur, Málaga, 29071, Spain
ORCID: 0000-0002-7779-8810
E-mail: lolaburgueno@uma.es

G. Liebel
Reykjavik University
Menntavegur 1, 102 Reykjavík, Iceland
ORCID: 0000-0002-3884-815X
E-mail: grischal@ru.is

the tools' interface and modelling knowledge. We observed how modelling tools influence students' modelling styles and how they can be used to foster students' confidence and creativity. Based on these observations, we developed a set of guidelines aimed at enhancing modelling education and helping students acquire practical modelling skills. **Conclusions:** The guidance for modelling in education needs to be structured and systematic. Our findings reveal that different modelling styles exist, which should be properly studied. It is essential to nurture the creative aspect of a modeller, particularly while they are still students. Therefore, selecting the right tool is important, and students should understand how a tool can influence their modelling style.

Keywords Software Modelling, Observation Study, UML, Education

1 Introduction

Software modelling holds significant promise for enhancing various aspects of software and systems engineering, such as productivity [3] and cost efficiency [27]. Despite these advantages, the widespread adoption of software modelling across the entire field remains limited [23]. Extensive research has explored the reasons behind the limited adoption, revealing issues like subpar code generation, inadequate tool support, and a lack of guidance or training [20,58,57,38,34,33]. Specifically, it has been suggested that engineers are reluctant to embrace modelling because it requires excessive effort and offers little usefulness, a view shaped by their educational background [23,53]. Within the educational domain, substantial efforts have been dedicated to understanding how to teach modelling within university curricula, exemplified by works such as [55,13,28,40,56,52,32]. This body of literature often takes the form of proposed course designs [51,55], experiential reports detailing challenges or best practices, particularly concerning tools [13,28,40,56], or papers presenting quantitative studies on student opinions [52,32]. Collectively, this wealth of knowledge serves as a valuable source of experience and inspiration for the effective teaching of modelling practices. However, research on students' perspectives on modelling needs to provide more detailed explanations, such as specific challenges and benefits regarding modelling assignments, modelling tools, syntax, and semantics. Furthermore, experience reports from researchers with a specific focus on software and systems modelling may run the risk of being unrepresentative for instructors lacking such a focus. Given that most students will transition into professional roles post-graduation, it becomes crucial to comprehend their perceptions. It is essential to understand the challenges students currently face and explore the process of modelling that students practice in universities.

In our prior research [15], we engaged in 16 interviews with students and 5 interviews with instructors to explore students' perceived understanding of modelling and the challenges they encountered. The findings of our study indicate that students recognise specific advantages of modelling, particularly in areas where informal models suffice, such as gaining a system overview or

conceptual comprehension of the problem domain. However, the majority of the students remains skeptical about the value of modelling in detailed system design. Many students expressed challenges related to issues such as selecting the appropriate notation, determining what aspects to express in the models, and understanding how to apply modelling techniques to unfamiliar domains.

In this paper, we dig deeper into students' modelling practices by undertaking an observational study. We conducted two modelling sessions where students participated in pairs and solved modelling tasks. Our goal here is to observe students' modelling actions and interactions during the modelling task and explore the challenges experienced by an individual or pair.

The study follows two research questions (RQs):

- RQ1: What actions do students take on a modelling tool to solve a task?
- RQ2: What challenges do students face while solving modelling tasks?

RQ1 focuses on the actions students undertake within a modelling tool as they solve a given task. Understanding the specific steps students employ in a modelling tool provides valuable insights into their cognitive processes and problem-solving strategies. RQ2 investigates the challenges students experience while engaging in modelling tasks, including challenges based on students' modelling perspectives, tool knowledge and the communicative dynamics between pairs of students engaged in collaborative problem-solving.

This paper presents findings from an observational study conducted with 32 students from two different universities, Reykjavik University, Iceland, and University of Malaga, Spain. The students worked in pairs, engaged in solving tasks using two modelling tools, namely MagicDraw[1] and PlantUML[2].

Our results show that students need more knowledge of structuring and completeness in modelling. While students are confident drawing class diagrams, sequence diagrams still pose a challenge to them. In both cases, students need to improve how they read and process the problem before modelling its solution. We observed how modelling tools influence students' modelling styles and can be used to provide students with the necessary confidence and creativity. Finally, we formed a set of guidelines based on these observations, which will help in modelling education and help students adhere to the practical skills of modelling. The rest of the paper is structured as follows.

In Section 2, we discuss the existing literature related to modelling education, students' modelling experience and observational studies. In Section 3, we describe the pilot studies, observational study setup, data collection and data analysis strategies. In Section 4, we mention the summarised results of the observational study, followed by a detailed discussion of the results in Section 5. Finally, we conclude our paper in Section 6.

[1] https://www.magicdraw.com/
[2] https://plantuml.com/

2 Related Work

Model-Driven Engineering (MDE) or Model-based Engineering (MBE) is the current state-of-the-art in software abstraction, where models are used as primary artifacts in the software engineering process. It has gained popularity in academia and industry for managing the complexity of modern software and is considered a significant advancement in software development [26]. Although the benefits of MBE are often considered obvious, higher abstraction levels do not guarantee better software and while code generation may boost productivity, the effort to develop models and make manual modifications can negate this benefit [25]. Regardless of the success of MBE, abstract models are crucial to the future of software development. When it comes to software modelling, UML (Unified Modelling Language) and UML-based development methods have become de facto standards in SE. Text books on UML and object-oriented design commonly refer to heuristics for creating UML diagrams, e.g., [50, 11, 1].

In the following, we discuss literature related to UML modelling in education, especially modelling done by students, modelling challenges and observational studies as a method of investigation.

2.1 Modelling in Education and Modelling by Students

Numerous studies have explored modelling in education, with a predominant focus on teaching modelling methodologies [55, 21, 22] or presenting experience reports from modelling courses [4, 40, 28, 12]. In [55], Westphal presented the design of a software engineering course heavily influenced by and focused on UML modelling. In [21], the authors proposed a course where students acted both as language designers and users to evaluate the usability of software language engineering. Gonnord et al. [22] took a reverse approach, guiding students from low-level C code to designing a modelling language workbench.

Akayama et al. [4] shared their experiences and opinions on tool use in software modelling education. The paper describes various approaches taken by the individual authors and discusses factors such as modelling tools versus pen and paper, the conflict between design and programming concepts, and methods to measure the quality of models. Paige et al. [40] discussed what they consider to be bad practices in teaching modelling. They identified issues such as covering too broad a range of modelling-related topics and focusing on syntax instead of semantics. Similarly, Kolovos and Cabot [28] presented a corpus of use cases for courses teaching model-based engineering (MBE). In [12], the authors discuss previous challenges with modelling encountered in various software engineering courses. They then present a case study in which students define a system and its views, simulate them, examine their relationships, and conduct several types of analyses on the complete system specifications. The authors also include a survey to know students' feedback.

While some surveys assess students' overall experiences with modelling [18, 2], there remains a scarcity of studies providing a detailed account of students' experiences in modelling or reporting results derived from students actively creating models or solving modelling problems. In the domain of business process modelling, studies like [42] have documented the modelling experiences of 115 students, highlighting three distinct modelling styles, "*We could distinguish (1) an "efficient modeling style" characterized by a limited time needed to think about the modeling task, and a fast rate of adding elements to the model; (2) a "layout-driven modeling style" which involves much time in creating a comprehensible layout while being less efficient in creating the model; and (3) an "intermediate modeling style" that is neither particularly efficient nor invests particularly into model layout".*

2.2 Modelling challenges by students

Multiple studies reported students' challenges regarding software modelling. Stikkolorum et al. [52] explore student difficulties and modelling strategies within UML Class diagrams by providing students with a specialized UML editor featuring feedback mechanisms. The findings identify four distinct strategies (depth-less, depth-first, breadth-first, and ad hoc), representing various approaches to constructing class diagrams based on task requirements, encompassing associations, attributes, and problem-solving elements. Reuter et al.'s research [48] delves into students' challenges with UML diagrams across two modelling courses, resulting in the compilation of a comprehensive catalogue of issues associated with UML diagrams. Agner et al. [2] broaden the scope with a survey on modelling tool use among 117 students, uncovering issues such as a lack of feedback, diagram-drawing difficulties, and tool complexity. In a different study, Hammouda et al. [24] compare the utilization of modelling tools to traditional pen-and-paper methods. Through a survey, they evaluate students' perceptions of the differences, ultimately finding no clear advantage for either approach.

2.3 Observational Studies in Literature

Observational studies are beneficial in investigating participants' thinking processes and actions. In their paper, Aniche et al. [6] observe 13 developers thinking aloud different test cases. Their findings show three different strategies used by developers while testing and explain the reason behind these strategies. Dekel et al. [19] perform an observational study of several collaborative design exercises. Their results show the popularity of the ad-hoc nature of collaborative design and the behaviours of different design teams in different contexts and used communication mediums. In their study, Mangano et al. [36] observe eight professional software designers doing 14 hours of design activities on whiteboards to analyse the communication between designers and

whiteboard sketching. Carver et al. report in their study [14] an observational approach where an experimental subject executing a procedure was observed by another individual. The study was conducted in pairs, with the participant implementing the technology referred to as the **executor**, and the individual observing the technology's application termed as the **observer**. Importantly, the role of the observer did not involve collaborative work with the executor in applying the technology; rather, their responsibility was to ensure that the executor adhered faithfully to the technology's procedural steps. The observer also took notes on the executor's utilization of the technology, highlighting instances where challenges were encountered. After a certain point, the roles of executor and observer were swapped. The authors claimed that this observational study technique provided a level of detail regarding individual process steps and their efficacy that is challenging to obtain through traditional post-experiment questionnaires. A similar strategy has been used in pair programming for ages. Pair programming [59] is becoming a popular practice in software development [60]. In their survey, Begel et al. investigate the advantages of pair programming [9], identifying benefits such as enhanced code understanding, increased creativity and brainstorming, and improved design.

Motivated by these insights, we embraced a pair programming-style observational study approach to achieve similar benefits in the context of modelling. Acknowledging modelling as an inherently creative task, especially for our predominantly first-year undergraduate participants, collaborative problem-solving emerges as a valuable asset. In contrast to the approach outlined in [14], in our study the **observer** assumes the role of a **navigator** who directed the **executor**. This shift in dynamics differs from the traditional **executor** role, to a **driver**, emphasizing the collaborative and interactive nature of our pair programming-inspired methodology.

3 Method

Observational studies constitute a cornerstone in scientific research, providing a valuable method for investigating and understanding natural phenomena in their real-world context. Unlike experimental studies, which involve deliberate manipulation of variables, observational studies involve the passive observation of subjects in their natural settings. For our study we followed the method proposed by Williams at al. [59,60] inspired by pair programming, which is a popular practice in software development. Begel et al. explore the benefits of pair programming in a survey [9]. They identified code understanding, creativity and brainstorming, and better design among the benefits of this method. We decided to follow a pair programming approach (in our case *pair modelling*) for similar advantages. We are also following constructive interaction [5,7,37] and collaborative learning [54] for data analysis. Furthermore, we are using the Empirical Standards for Software Engineering Research [46][3]. In particular we

[3] `https://www2.sigsoft.org/EmpiricalStandards/tools/`

are using author checklist provided in the *General Standard* category, which contains the specific criteria that can be used by authors to conduct and report research. Understanding and solving a problem with software modelling is a creative task. Modellers read the problem from their perspective and conclude the solution based on individual thinking and knowledge of the domain. The modellers' style, apart from modelling tools and notations, influences the solution. There are no significant studies in Software Engineering (SE) regarding software modelling style. A few studies in business process modelling (e.g., [43]) lead the way towards learning about style.

Our study mainly focuses on modelling actions, like *Add, Delete, Edit*, and challenges to understanding students' modelling process. We detail our study steps in the following subsections, from the pilot to the final study setup.

3.1 Pilot Study 1

For pilot 1, we utilised the tool Papyrus and a plug-in extension called ModRec [45] to create the models and gather data, respectively. With ModRec, modelling actions can be automatically captured and can be viewed as a separated file. The study was promoted to undergraduate software engineering students at Reykjavik University (RU), resulting in three volunteers. Pilot 1 was conducted in September, 2022. All three participants were briefed on the pilot's objectives and had the option to withdraw at any time. They did not receive any compensation for their participation. It was also clarified that their performance would not have any impact on their grades, neither for good or bad. All three were first-year students familiar with UML modelling.

The students were tasked with drawing a class diagram to depict the domain of a restaurant food court system. They were instructed to use Eclipse Papyrus as their modelling tool and were given a demonstration of the tool's interface and the ModRec plug-in. Several requirements were provided, and they were allotted 30 minutes to complete the diagram. The details of the problem description used in Pilot 1 can be found in the Appendix A. We observed students facing difficulties with Papyrus, realising they invested most of their time understanding the tool instead of focusing on the task. Additionally, we realised that the example domain, a restaurant food court, was not interesting enough to students.

Based on these observations, we made three changes: (i) using a more relevant and exciting problem domain, (ii) adding not just structural, but behavioral aspects to the modelling problem and (iii) not prescribing any modelling tool, letting the participants select one themselves.

3.2 Pilot Study 2

After integrating in our study the changes from the pilot study 1, we selected a dating app as the problem domain (more details below) and created two

tasks involving drawing a class diagram and sequence diagrams. We introduced structural and behavioural modelling in the pilot 2 to get more details about the students' modelling process. The first task was to draw a class diagram to capture the app's domain. The second task was to draw two sequence diagrams to model two dating app functions.

We select the dating app as our domain to get students interested in solving the problem. Our study is based on student volunteers, and getting their attention towards the tasks is necessary. We looked through past courses and realised that it is not a traditional problem domain, and students might be curious to attempt it. Also, we decided to go with one static and one dynamic UML diagram type. Prior experience [15] suggests that class diagrams are the easiest UML diagram type for students to understand, and sequence diagrams are the hardest. Lopes et al. [35] find that students find sequence diagrams not easy to use alone, as they need other diagrams, such as use case and class diagrams. In [47], Reuter et al. survey students' problems with UML diagrams. Regarding sequence diagrams, the authors find students need help with the order of elements, actions in a sequence diagram, needing help to identify interacting classes that are needed as objects for the sequence diagram, etc. Inspired by these studies, we decided to use class and sequence diagrams. Since we let students use their preferred modelling tool, we decided to record their screens to get the modelling actions following a method equivalent to pair programming as stated above [59,60].

For the first problem, student A plays the driver, and student B be the navigator. That means student A solves the problem by drawing diagrams, while student B navigates through the problem, giving insights and inspecting the drawing. For the second problem, students switched roles. Also, we thought by pairing up, future participants might feel more motivated to join the study. We are aware of the shortcomings of pair programming, such as personality clashes, and bad communication affecting the design. However, we wanted to see how communication between a pair influences the design.

The study was divided into four sections: Reading the full problem description (10 minutes), one of the participants solving task 1 with a class diagram (20 minutes), taking a small break (10 minutes), the other participant solving task 2 with sequence diagrams (20 minutes). The details of the problem description used in Pilot 2 can be found in the Appendix A. We then conducted the pilot study 2 in November, 2022 with two PhD students of the computer science department at RU in . Both PhD students had prior experience of UML. The participation was voluntary, they had the option to withdraw at any time and did not receive any compensation for their participation. The two volunteer PhD students selected a tool of their own (draw.io), based on their previous experience of the tool.

After observing the second pilot study, we decided to change the time frame and a few details in the problem description used for the study. We reduced the time for reading the problem and break to 5 minutes each and added 5 minutes each to the problem-solving. We realised that giving participants an open problem description is not a good idea for the problem description, as

the two volunteers in the second pilot study took a safe path and created generic diagrams. Hence, for the main study, we added listed requirements to our problem description, which is discussed in the next section. For the main study, we recruited volunteers from two universities (11 pairs from University of Malaga, Spain, and 5 pairs from RU, Iceland).

In the next three sections, we discuss the study design and setup (Section 3.3), data collection (Section 3.4) and data analysis (Section 3.5).

3.3 Study Design and Study Setup

For the final study, we contacted the University of Malaga (UMA) in Spain and RU in Iceland. In UMA, we considered Software modelling and design, a bachelor course in 3rd year with 110 students. The course is about structural modelling with class diagrams and behavioural modelling with sequence diagrams and state machines. It is taught how to model the transition of a system over time. Design patterns and the conversion of class diagrams into object-oriented code are also studied. We advertised the study in the course. One of the authors of this paper (a lecturer in the course) presented details of the study and research goals earlier in the course. Later, we asked for voluntary student participants, and 11 pairs signed up for the study.

In RU, we considered a bachelor course in the second semester of the Software Engineering program, with 51 students. The course centres on analysing, modelling, and designing software systems, employing an object-oriented approach and utilising UML. Throughout the course, students learn class, sequence, state machine, activity, and component diagrams. Additionally, the curriculum includes the practical implementation of the designed systems using the Python programming language. We made the study the last assignment in the course. Five pairs volunteered for the study. Thus, we asked them to follow the study setup. The rest of the students finish the problem as individual assignments for the course.

For the final study, we changed the problem description from pilot study 2 and added listed requirements to give students more specific details and restrictions about the problem domain. Then, we kept the tasks similar to pilot 2. The details of the final problem description can be found in the Appendix A. Students signed up in pairs, selecting their own partners. In UMA, participation was voluntary, and participants could withdraw from the study at any time. Students were orally informed about the purpose of the study: conducting a research experiment. They knew and agreed that their data would be anonymised and the results could be part of research publications. The task was undertaken near the end of the semester, in mid-December 2022. Students in RU signed a consent form before participating in the study. Participation was voluntary, and participants may withdraw from the study at any time, without penalty. Furthermore, participants could deny the researchers conducting the study the right to use the collected data (from their participation). Since the task was part of the course assignment, participants received grades

as per course rules and feedback on their modelling solutions. The study was conducted in April 2023.

Each pair got one laptop and a recorder to record their conversations, and we also asked them to use screen recording so we could record their modelling activities. In UMA, students used Magicdraw as they were familiar with that tool. In RU, students used PlantUML as it was the tool used in that course. Students received the problem description in English. To get their authentic reaction, make them feel comfortable and avoid language barriers, we gave them the option to converse in their mother tongue or in English. All 11 pairs from UMA conversed in Spanish, three out of five pairs in RU spoke Icelandic, and two spoke English.

3.4 Data Collection

11 pairs from Malaga participated in the study in Spain. Similarly, five pairs from Reykjavik participated in the study at the end of their course in Iceland. In Malaga, the study was conducted in December 2022, and four months later, in April 2023, the study was conducted in Iceland. After a brief analysis of the data collected from all 16 pairs and an extensive analysis with two pairs each from UMA and RU, we concluded the study. We observed a sufficient similarity in the data, indicating a saturation point [8]. The repetition of data suggested that further collection would be redundant, confirming that we had reached an adequate sample size. Also, we found sufficient diversity, depth and probable issues demonstrating validity of the collected data.

Students had 5 minutes to read the problem description, 25 minutes to solve the first problem (class diagram), a break of 5 minutes and then 25 minutes to solve the second problem (sequence diagram). We collected three forms of data: (i) *modelling actions through screen recordings*, (ii) *voice recordings of the conversations between a pair*, and (iii) *PDFs of each pairs' final diagrams*. The three forms of data address our RQs directly. The screen recordings summarise actions taken by students while modelling. All three data sets explain the challenges students face while modelling.

3.5 Data Analysis

Initially, we classified the recordings (both screen and voice) into three sections: First, reading the problem during the start of the study when a pair reads out the problem (5 minutes); then, drawing the class diagram with a time limit of 25 minutes; finally, drawing two sequence diagrams, also with a time limit of 25 minutes. We further classified each of these sections into three categories based on our RQs: actions (doing modelling in terms of Adding, Deleting and Editing the diagram), communication (planning modelling) and challenges (about modelling). While analysing actions, we considered data that describes any modelling action taken by a pair in the modelling tool, e.g., Adding a class,

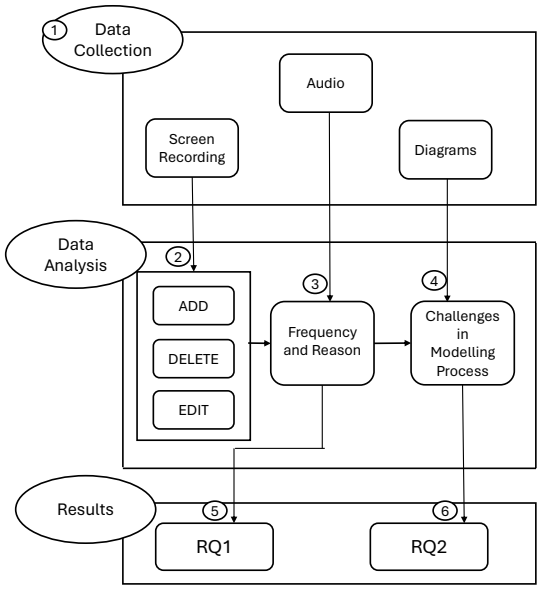


Fig. 1: Data Analysis

Deleting an attribute, or Adding a connection. We also included timestamps for each of these actions. For communication, we included dialogues between the pair and their timestamps. These dialogues covered agreement or disagreements between a pair, decision-making points, or conversations leading to an action. Finally, challenges include any piece of conversation between a pair or action that describes challenges a pair faces while modelling or about modelling.

We analysed four pairs (2 from UMA and 2 from RU) with the abovementioned strategy. However, we realised distributing the data like this was tiresome, and threats of researcher bias were increasing. Hence, we made a more compact and precise protocol. We focused on the class and sequence diagrams this time, precisely how a pair approached and solved those modelling tasks.

Figure 1 describes the analysis strategy. We dedicated the first step towards data collection. Next we break down students modelling into three distinct actions, *Add, Delete* and *Edit*, based on the screen recordings. An *Add* action occurs when students incorporate new elements into their model, ranging from classes, attributes, and methods to associations between classes, participants, and messages—both direct and return. Conversely, a *Delete* action is logged

when students remove any component from their models. Lastly, an *Edit* action encompasses modifications to existing *Add* actions. This involves alterations such as changing the name of a class or message, adjusting multiplicities, and redirecting associations between classes. These defined actions provide a comprehensive framework for analyzing the dynamic evolution of students' modelling throughout the study.

Then, we focused on each action and incorporated the audio feature in the analysis. That is, we tried to understand the frequency of those actions and the reasons behind their occurrence. Finally, we then looked at students' whole modelling process and the final diagrams to understand what type of challenges students experienced while drawing the diagrams.

Similar to our previous approach, we selected two pairs from each university, performed the analysis, and discussed among the four authors. When we reached an agreement, the first author conducted the rest of the data analysis. Based on our complete analysis, we answered the RQs (step 5 and 6). We report them in detail in the next section (Section 4).

3.6 Threats to Validity

The results in this paper are based on observing 16 pairs of students across two universities in two different countries. The collected data was in three different languages. We identified several threats to validity, which we describe below along with the measures we took to mitigate them, classifying the validity threats in three categories as described in [61].

3.6.1 Internal validity

Internal validity focuses on how confident we can be that the elements of the study actually caused the observed outcome. There may be other factors influencing the outcome, which are beyond our control or have not been measured.

One of the threats in our study is that the students are Icelandic and Spanish, while the problem description was in English. To mitigate this threat, one of the authors, who is Spanish, was present during the study at UMA so that students could clear their doubts. In RU, the original course instructor, who is Icelandic, helped students with questions.

3.6.2 External Validity

External validity is concerned with whether we can generalise the results beyond the scope of our study.

Our study participants are all bachelor students, and while doing the study, they were active in a course which involved software modelling with UML. Two threats here are the modelling background of study participants and the use of a UML diagram in the study. For our study setting, it was essential to have a similar modelling knowledge among participants, which was achieved. We

used two appropriate modelling diagrams, class and sequence, in the study settings.

Despite all participants are bachelor students, we believe this threat is minimised because students were active in a course involving software modelling with UML, and they had been doing similar exercises during the course, so all the concepts and knowledge were fresh in their minds. Therefore, we believe the same study involving software engineers might have had similar results. This qualitative study, involving 32 individuals, provides valuable insights that can be highly relevant and applicable to similar contexts. While claiming generalisability may not be appropriate, the results offer a strong foundation for understanding the phenomena studied and can potentially inform practices in other comparable settings.

3.6.3 Conclusion Validity

Conclusion validity focuses on how confident we can be that the study elements and settings are genuinely related to the observed outcome and whether the study can be repeated.

We kept a flexible study setting for our student participants as we wanted to capture their interactions with each other and with modelling. The study occurred in their university rooms, where students used their preferred modelling tool and modelled with their language. Conclusion validity refers to the extent to which the conclusions drawn from a study are credible and justifiable. It ensures that the relationships identified between variables are genuine and not influenced by external factors or biases. While we cannot entirely eliminate the presence of researcher bias in our results, we implemented multiple measures to mitigate this threat as much as possible. For data analysis, we began with a smaller sample size from our data (two pairs from each university) and had all four authors analyse the data. We then discussed our findings among ourselves, and once we reached an agreement, we analysed the rest of the data accordingly. Consistency was maintained throughout our study design, data collection, and analysis between the two universities. For analysing the screen recordings, we used the terms Add, Delete, and Edit, with all four authors defining and agreeing on these terms. Voice recordings presented a challenge due to the involvement of three languages. However, we transcribed the conversations into English and used live coding to capture the students' perspectives accurately.

4 Results

We analysed 16 student pairs from two universities as they solved a modelling task by drawing class and sequence diagrams using two modelling tools. We observed several patterns in the actions taken and formulated different challenges faced by the pairs to solve the modelling problem. Below, we answer our two research questions using the collected data.

4.1 RQ1: What actions do students take on a modelling tool to solve a task?

To answer RQ1, we analysed students' screen recordings and distinguished patterns to draw different diagrams. These patterns are on a micro level, detailing how students modelled each element of these UML diagrams. Additionally, the patterns observed are specific with respect to class and sequence diagrams and highlight the individuality of students while drawing each type of diagram. Combining all the patterns observed from students while drawing both class and sequence diagrams reflects the diverse modelling styles among them. In our previous study [17], we defined modelling style as **"consistent individual differences in preferred ways of creating and processing software models"**. The individual preferences displayed while drawing both diagrams revealed hints of distinct modelling styles. In their study, Pinggera et al. [42] conducted two large-scale modelling sessions involving 115 students. Their study is similar to ours in terms of recording modelling sessions and considering Add, Edit, and Delete as three distinct modelling actions. After cluster analysis, they reported three distinct modelling styles for business process modelling. We recognise that it is premature to categorise the actions observed in our study as styles based on just two diagrams and 32 students. But unlike Pinggera et al. [42], we recorded students conversation as well, which gave us more depth into their modelling actions. Therefore, we decided to answer RQ1 by presenting these patterns as *"modelling preferences"*, which can later be examined with more diagrams and a wider range of problems.

4.1.1 Modelling preferences for class diagram

Students employ diverse strategies when it comes to completing a class with all its attributes and methods, and we observed *three* distinct preferences. *Firstly*, some opt for empty classes, primarily following the "noun identification" method. They identify all the necessary classes for the diagram, draw them, and subsequently add attributes and methods. The *second* category involves students starting with the identification of one class at a time, gradually adding basic attributes before progressing to the next class. Lastly, the *third* preference involves a mixed method, where students begin by creating all the classes, then move back and forth between them, adding attributes and methods.

By combining the audio with these actions, we observed that students visualised the problem and mapped to its solution in a way that made constructing classes and their relationships appear disorganised. This might be influenced by their experience and familiarity with the diagram style. However, while these factors may play a role, the reason is also related to their personal style. A notable piece of evidence is the difference between the two datasets from the two universities. At UMA, the students were in their third year, while at RU, the students were in their first year. We observed the third approach to drawing classes in four out of the five pairs at RU, which might be due to their inexperience. At UMA, the distribution of the three approaches was

almost equal. Hence, we can assume this variation is due to personal style, with individual preferences for moving back and forth between problem and solution, rather than completing a class fully. Instead, they try to complete different pieces of the whole problem and establish the solution in that way.

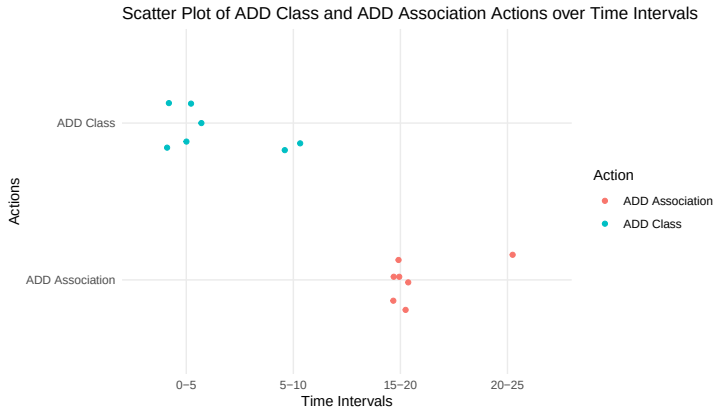
Students adopt *two* distinct preferences when connecting classes in their class diagrams. The *first* preference involves drawing all classes initially and establishing connections later, while the *second* preference entails sequentially drawing classes and their associations. Variations in the second preference include drawing two classes before connecting them and drawing one class, adding an association, drawing a second class, and then connecting them.

Notably, all five pairs using PlantUML, a text-based modelling tool, follow the first preference. In contrast, all 11 pairs using MagicDraw, a more visually-oriented tool, favour the second preference. Hence, it seems the tool significantly influences these choices. PlantUML's code-based nature encourages the construction of larger blocks (classes) first for efficiency, as students can view the generated model after running the code. In contrast, MagicDraw's graphical interface provides visual feedback from the start, with associations drawn by dragging arrows or lines, offering a better visual experience compared to PlantUML. Figure 2 shows two different behaviours seen in the two modelling tools. In Figure 2a, we can see two actions, Add Class and Add Association, from two pairs of RU students. The figure clearly shows how the Add class actions happened initially, followed by the Add association action. In Figure 2b, we see a different picture; with the same two actions and two pairs from UMA, we can see the actions are intertwined.

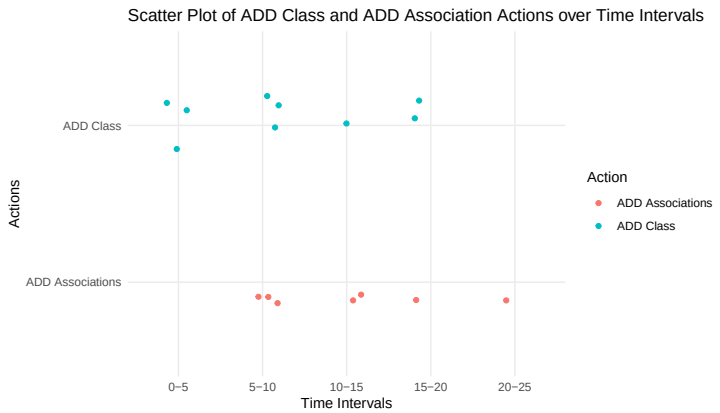
4.1.2 Modelling preferences for sequence diagram

In contrast to class diagrams, students often modified their sequence diagrams. Hence, the emphasis was more on the Delete and Edit actions. Students exhibited a higher frequency of Delete actions when working on sequence diagrams, a trend consistently observed across both tools. The increased use of Delete actions suggests a notable pattern in the modification and refinement of sequence diagrams during the modelling process. After analysing the videos, we concluded the following purposes for using a higher amount of Delete actions in sequence diagrams:

1. Communication Issues: students were modelling in pairs and following an approach similar to pair programming. We found the *driver* deleting some elements as there was a misunderstanding or miscommunication between the *driver* and the *navigator*.
2. Wrong Syntax/Tool issues: the *driver* was realising syntax mistakes and deleting an element, heavily influenced by the tool's interface. When utilising PlantUML, we noted a frequent reliance on the "copy-paste" function by students due to its text-based nature. This increased use of the "copy-paste" function consequently resulted in a higher frequency of Delete and Edit actions during the modelling process, as pasted content had to be adapted.



(a) Connecting classes in PlantUML



(b) Connecting classes in MagicDraw

Fig. 2: Different ways to connect classes with in PlantUML and MagicDraw

3. Brainstorming: Students deleted elements from the diagram while brainstorming the solution. Students would Add a tentative element, e.g., a class name or an association, to have a basis for discussions. These concepts would then frequently be Edited or Deleted again while the discussion was ongoing.

While reviewing the videos, we encountered situations where Delete and Edit actions were back-to-back, necessitating a clear distinction between these two

actions. It is important to note that, in class diagrams, we did not observe this pattern, indicating that students are more confident about modelling the domain with class diagrams.

We observed inconsistent preferences or strategies across the sequence diagrams, with students employing varied message types throughout their drawings. We believe this happened because they did not have a clear method, like the "noun identification" method, to follow. In Figure 3, we can see two sequence diagrams for creating a profile in the dating app with appropriate information and two different diagrams with different strategies. In Figure 3a, students used three objects, i.e., *Account*, *Profile* and *Verify*, to demonstrate profile creation with verification in the dating app. For the same problem, the second pair used two objects *DatingApp* and *BackEnd* in Figure 3b. Note that both pairs used different strategies to fulfil the requirements, like feeding and verifying information into the profile. Additionally, while the first pair used one direct message line to feed all the personal information into the profile, the second used separate message lines. The problem description listed all the personal information needed to create a profile. Notice the difference between the two pairs and how they read the problem.

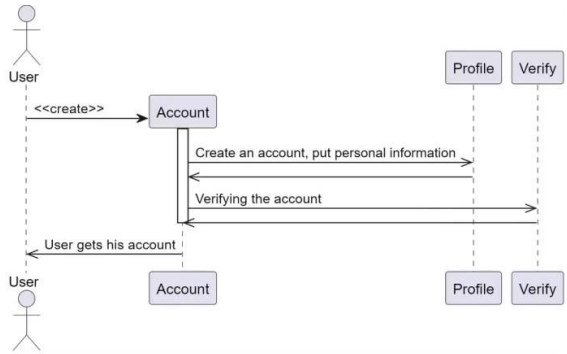
4.2 RQ2: What challenges do students face while solving modelling tasks?

To answer RQ2, we integrated the modelling actions with the interactions between students while solving the modelling problem. We observed three main challenges that we detail in the following.

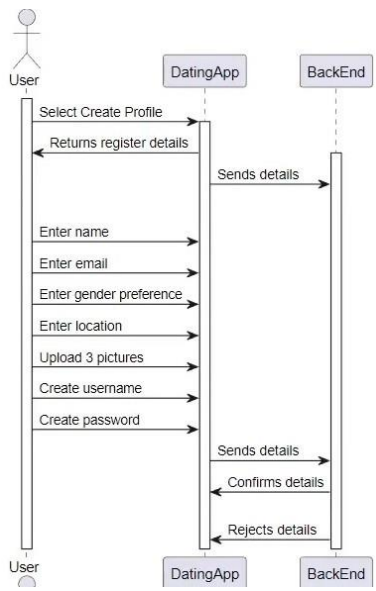
4.2.1 Inconsistent strategies across different diagrams

One of the challenges we observed is the use of inconsistent strategies while solving the modelling problem. This issue was less evident in class diagrams. Although we observed different strategies for creating a class diagram, they were not particularly disorganised. However, in the sequence diagrams, students used varying terms to describe messages and objects.

One reason behind this inconsistency is how students interpret the modelling problem. In our previous studies [15, 16], we noted that there is no systematic method taught to students for practising the reading of a modelling problem, which is a significant issue. The consequence of this challenge is that students' understanding of the correct diagram can be misleading. As seen in Figures 3a and 3b, two different diagrams from RU show the same functionality of creating a profile in the app, but each of them employs a different design. A similar situation happens in the diagrams in Figure 4 from UMA, where we can see that different students go into different levels of detail in the diagrams. We understand that, since modelling is a creative task, having different solutions is predictable. However, guidance is needed to explain to students why they are modelling in different ways.

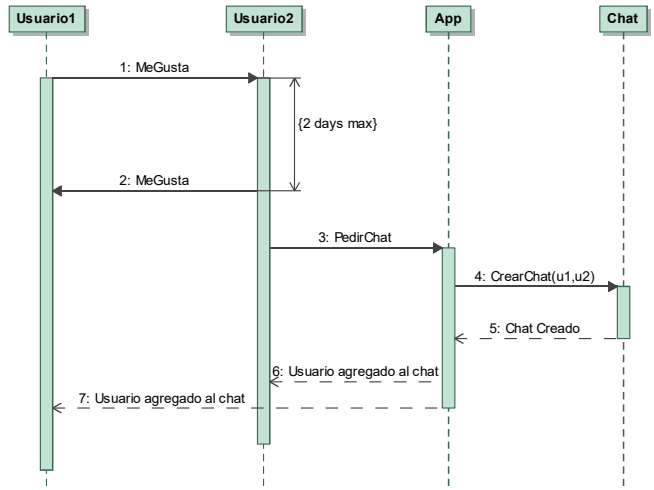


(a) Pair using one direct message line to feed information into the profile

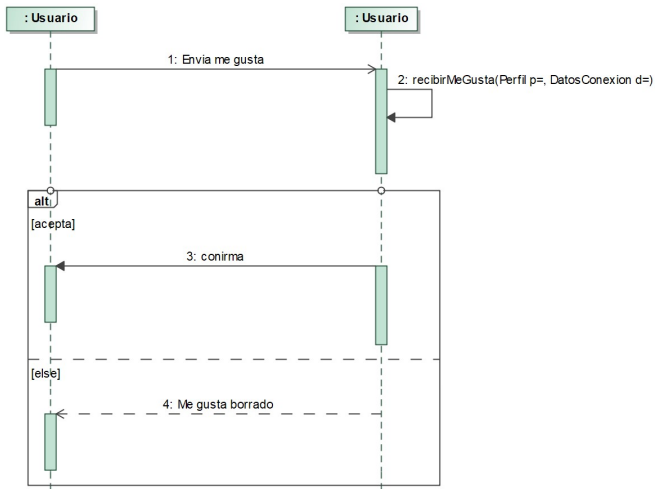


(b) Pair using several direct message lines to feed information into the profile

Fig. 3: Sequence Diagrams from RU awith different strategies for profile creation



(a) Pair considering the like and the creation of the chat



(b) Pair only considering the likes

Fig. 4: Sequence Diagrams from UMA about giving a “like” to someone

4.2.2 Challenges in Interpreting a Modelling Problem

Our study setting allowed students to read the problem for five minutes before beginning the modelling task. Although students had both tasks (class and sequence diagrams) from the start, the majority focused on reading the problem to draw the class diagram first. Additionally, a common pattern we observed is that students preferred reading the problem description to solve the class diagram and then referred to the class diagram to solve the sequence diagram. Consequently, if they missed something in the class diagram, the subsequent sequence diagram was affected. Students realised the importance of reading the problem thoroughly to solve both tasks only later, while finishing the sequence diagram, and they acknowledged their oversight.

> *"N: Let's not do that. We should have fixed that in class diagram, I think it was mentioned in the problem"* — RU_Student[4]

> *"D: But how can I move that though? Like maybe we have to deactivate. N: Should read first"* — RU_Students

> *"D: Can we change the class diagram at this stage (while realising the sequence diagram)? N: I don't think so"* — UMA_Students

> *"D: We should have spent more efforts when depicting the class diagram. N: I agree"* — UMA_Students

> *"D: I think the thread should have been a class. N: I agree, but that's already done, now we cannot leave it like this in the sequence diagram."* — UMA_Students

Following a conversation piece between a pair while solving a sequence diagram task where students had to model the requirements, *a user can "like" another user if they share the same geographic location, a user has two days to confirm or decline a "like" from another user"*. This piece of conversation shows struggles and lack of systematic approach to read a modelling problem/model.

> *"N: It's not like this, because this arrow indicates User1. So maybe we have to put another arrow from User1 to User2 or something. D: No, no, it doesn't matter. N: Wait, what if both share the same? Oh, no, no. I was reading this wrong. D: So okay, so we get a notification we can skip the arrow that goes back to user one, right? N: Oh no no, they don't share the same location type"* — RU_Students

The challenge here highlights students' lack of training in reading problems and diagrams. Having an unsure attempt while drawing such a diagram is understandable. However, a structured way of reading a problem needs to be added here, furthering a method to create this model systematically. We can see students employing both approaches to solve a modelling task, but due to insufficient guidance, they struggle with each.

[4] *D* and *N* refer to *Driver* and *Navigator*, respectively

4.2.3 Lack of knowledge in the completeness of a diagram

Students commonly face uncertainty regarding task completion, experiencing doubt after both class and sequence diagram assignments. The challenge lies in the difficulty students encounter in verifying the accuracy of their modelling steps, leading to uncertainty about whether the diagrams effectively address the given tasks. The absence of a clear validation mechanism during the drawing process contributes to this post-drawing doubt among students.

> *"D: Whatever. We are done, I hope! We just have to fix this later, maybe. N: I think, this is alright(?)"* — RU_Student

> *"D: I don't think there is much more we can do."* — UMA_Student

> *"D: There's only 50 seconds left. N: Yeah, I'm not sure about the delete action, let's just leave it like this"* — UMA_Students

5 Discussion

In this section, we discuss our findings, particularly the issues identified through our results and their implications for modelling education.

5.1 Different Modelling Preferences

Stikkolorum et al. [52] conducted a similar study with 20 bachelor's students in software engineering during their second semester. Unlike our study, these students were inexperienced with UML modelling and only created class diagrams having learned class diagramming prior to the task. The authors recorded four activities from their class diagram drawing: create, move, set, and remove. They used their own tool, WebUML1, to log these activities. The authors report four strategies for drawing class diagrams in [52]. We found similar preferences for class diagrams among our students. However, our study, supported by students' conversations and the use of two different tools, provides a more in-depth picture behind these strategies. Three factors influence students' preferences: the modelling tool, the way they read the problem, and their personal style. We observed two main preferences for connecting different classes in the diagram. The first preference involves drawing all classes initially and establishing connections later, while the second preference entails sequentially drawing classes and their associations. Notably, all five pairs using PlantUML, a text-based modelling tool, follow the first preference. In contrast, all eleven pairs using MagicDraw, a more visually-oriented tool, favour the second preference. This strongly suggests that the tool has a significant influence on these choices. Our study also involves sequence diagrams, which helped us notice different preferences among students based on how they read the problem or the class diagram prior to modelling the sequence diagram. Additionally, including sequence diagrams shows the difficulties students have in connecting multiple UML diagrams, e.g., by using class diagram elements in their sequence

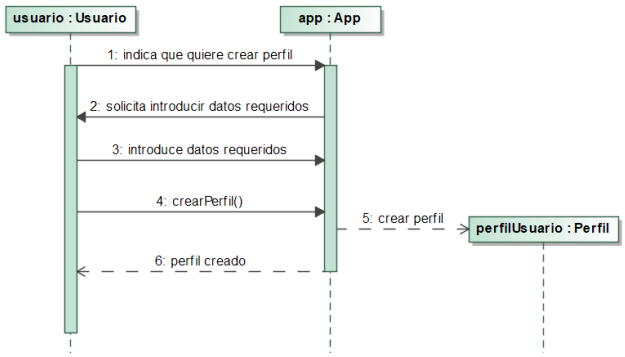
diagrams. Finally, since we employed a method similar to pair programming, where students alternated their roles, we observed variations in personal style. Some students favoured a more compact layout for the sequence diagram, while many followed the exact wording provided in the problem description, and others chose their own vocabulary. Unlike the class diagram, there was no noun identification method for the sequence diagram. In Figure 5, we can see two different sequence diagrams for creating a profile in the app, drawn by students from UMA using MagicDraw. The pair in Figure 5a mostly writes full sentences for the messages, while the pair in Figure 5b tries to follow the camel case notation and write what could be operation names—although such operations were not added in the class diagram.

Consequently, individual thinking influenced certain preferences among the students using different tools. In PlantUML, copying and pasting was a common trend, and during brainstorming, we observed that students often paused their actions. In contrast, students using MagicDraw frequently utilised many edit and delete actions, playing around with the interface by dragging parts of the diagram, especially in the sequence diagrams. An interesting question for future work is whether any of these approaches is more or less suitable for various modelling tasks, such as design exploration.

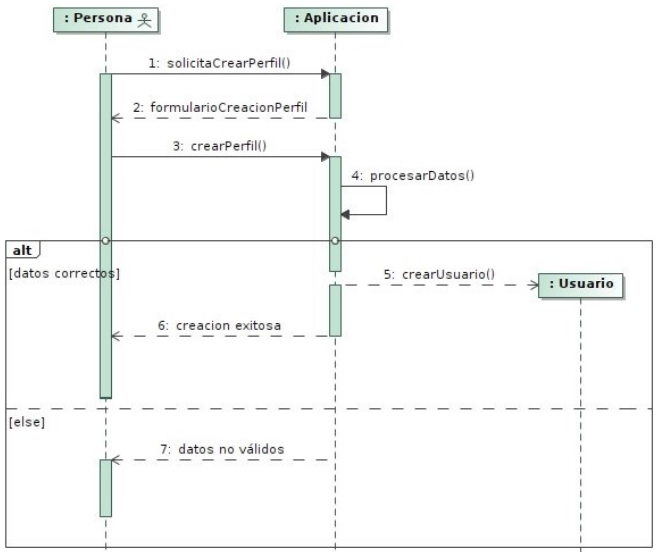
5.2 Structural vs Behavioural Modelling

Students find class diagrams relatively easy to model. Two primary reasons may explain this: the problem description explicitly stated "model the domain with a class diagram", which is less complex than modelling two functionalities with a sequence diagram. Another significant reason is that students find more relevance in class diagrams, which they can better visualise in relation to programming. For instance, in RU, the course instructor demonstrated this relevance using programming in lectures. Students often struggle to understand and model the behaviour of the system, finding it more challenging than creating the structural components. The class diagram provides the basic building blocks, which the sequence diagram then has to adhere to. In theory, one could start by designing all the interactions first, making the subsequent class diagram more difficult. However, in practice, students do not start with a "clean sheet" when working on sequence diagrams; they frequently refer to the class diagram to construct the sequence diagram.

Various studies [47,35] have addressed challenges with UML diagrams, particularly highlighting the difficulties associated with sequence diagrams. One study mentioned that students find it hard to imagine the sequence of events. In our previous research [15,16], students reported that sequence diagrams were challenging and the sense of a complete diagram was vague. We observed a similar trend in this study; conversations indicated that students felt more confident about finishing the class diagram than the sequence diagram.



(a) Pair writing mostly sentences in messages



(b) Pair trying to use camel case notation for messages

Fig. 5: Sequence Diagrams for creating a profile in the app (UMA)

5.3 Influence of Modelling Tools

Tool-related challenges are not new in software modelling education [32, 47]. The various designs and interfaces of modelling tools have created complexities in learning modelling. Pourali et al.[44] show several challenges modellers faced while modelling class and state machine diagrams with eight different modelling tools. Our study shows that the tools additionally influence a modeller's modelling style. Our study involved two different modelling tools: MagicDraw and PlantUML. While MagicDraw has a more "drag and draw" functionality, PlanUML is code-based. The distinct nature of these two tools influenced our students to follow a certain pattern during modelling. While drawing a class diagram, students using PlantUML created all the classes first and then connected them with the necessary associations. The reason is that, in PlantUML, one needs to run the code in order to get the visual product, and students found it helpful to see all the classes on the screen before completing the connection between them. In MagicDraw, the tool interface allows one to draw diagrams by selecting elements from the palette and dragging them to the main panel. Since visually, students could see the model in the making, students chose to complete a small part of the class diagram by creating connections between two classes and then moving forward.

In sequence diagrams, we observed more modification frequency than in class diagrams, i.e., more Delete and Edit actions. The tool's interface influenced here, too. Since PlantUML is code-based, students used a lot of "copy-paste" and, hence, used Edit actions. Not having consistent terminologies in sequence diagrams and the code-based nature of PlantUML made students more prone to errors. Comparatively, with Magicdraw, there were more Delete actions, as deleting an element or a thread of messages was easy with a click.

5.4 Implication to Education

Based on our findings, we emphasise the importance of training students to interpret a modelling problem and a model, to understand the correctness of a model, and to use modelling tools to foster creativity and an individual style. Before teaching modelling, it is crucial to incorporate instruction on interpreting both a modelling problem and a model. This practice does not only provide students with a foundational understanding of the modelling concept, but also instils a sense of correctness in their approach. Training students to read a modelling problem helps them gather the necessary information and map a mental model based on the problem and chosen diagram type. Studies in the past have demonstrated the difficulties of interpreting models. Studies like Kuzniarz et al. [30] and Zayana et al. [62] explore the involvement of stereotypical examples to enhance the comprehension of UML models. Kutar et al. [29] investigate whether users' understanding of information in sequence diagrams differs from their understanding of collaboration diagrams. We have observed students reading the class diagram to model the sequence

diagram, highlighting the necessity of an individual's ability to comprehend a model. When students read a model, they can understand the reverse journey from creation to the problem, identifying mistakes reflected in the model. This knowledge becomes invaluable when students create their own models. Additionally, reading a model imparts valuable insights into the visual aspects of modelling, helping students understand what the end-user can and should infer from the model. Consequently, students develop a nuanced comprehension of the information conveyed through the model, fostering a more comprehensive and insightful modelling process.

Unlike programming, modelling does not provide immediate error feedback. To address this, students should understand when their model is complete and correct. By correct, we mean that the resulting model fulfils all the criteria mentioned in the problem description. A potential approach is to teach students how to interpret a modelling problem into specific modelling concerns [31]. Figure 6 shows five modelling concerns that should be considered while modelling. In the early stages, understanding the objective and purpose is essential. These concerns also assist in interpreting an existing model. For educators, this implies that these modelling concerns need to be explicit, or exploring the concerns needs to be part of the task, as the solution space otherwise grows.

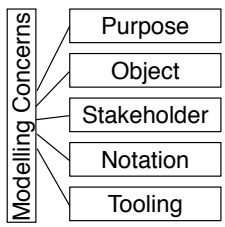


Fig. 6: Classification of Modelling Concerns. Adapted from [31].

One of the critical revelations from our results is that students' approaches to modelling vary based on the tools they use. Modelling tools are often considered challenging by students [39, 44]. However, beyond this challenge, modelling tools also shape students' modelling actions. Studies like [10, 41, 49] have investigated and explored different ways a tool can assist modellers; however, these are industry-level studies and do not focus on training students in modelling education. Additionally, the selection of a modelling tool is often left to the students. We believe it is beneficial to let students choose their own modelling tool, but we strongly recommend demonstrating the different interfaces

of several modelling tools and allowing students to explore these tools with different assignments.

6 Conclusion

Our study offers insights into students' actions and interactions while solving a modelling problem.

We conducted a modelling study with 32 students from two universities. The study involved solving a modelling task that included two UML diagrams: class and sequence. Students participated in the study in pairs, and we recorded their screen activities and conversations. After analysis, we observed several modelling preferences and challenges among the participants. These observations revealed modelling preferences that could evolve into distinct modelling styles with more detailed and broader investigation. By examining students' modelling actions alongside their descriptions of the modelling experience, we gained a deeper understanding of modelling as a creative task and identified several issues in modelling education that hinder its later adoption. These issues include inconsistent strategies, a lack of training in interpreting modelling problems, and the absence of methods to verify one's final model. We strongly recommend incorporating guidance into modelling education for interpreting modelling problems and creating models to address these issues effectively. The modelling preferences found through our study are promising and encourage further studies with different diagrams and a diverse range of problems. The results from these studies, in addition to the findings presented here, can lay the groundwork for identifying individual modelling styles, thereby guiding individuals towards more informative modelling practices.

Acknowledgement

We would like to thank all the students who participated in the study.

Declarations

Competing Interests

The authors declare that they have no conflict of interest.

Data Availability

We publish the problem descriptions we used for pilot studies 1 and 2 and the main study in the Appendix. We didn't disclose the videos of students modelling for privacy reasons.

References

1. Russell J. Abbott. Program design by informal english descriptions. *Commun. ACM*, 26(11):882–894, 1983.
2. Luciane T. Agner, Timothy C. Lethbridge, and Inali W. Soares. Student experience with software modeling tools. *Software and Systems Modeling*, 18(5):3025–3047, oct 2019.
3. Luciane Telinski Wiedermann Agner, Inali Wisniewski Soares, Paulo Cézar Stadzisz, and Jean Marcelo Simão. A brazilian survey on UML and model-driven practices for embedded software development. *Journal of Systems and Software*, 86(4):997–1005, 2013. SI : Software Engineering in Brazil: Retrospective and Prospective Views.
4. Seiko Akayama, Birgit Demuth, Timothy C Lethbridge, Marion Scholz, Perdita Stevens, and Dave R Stikkolorum. Tool use in software modelling education. In *EduSymp@ MoDELS*, 2013.
5. Jerry Andriessen, Michael Baker, and Dan Suthers. Argumentation, computer support, and the educational context of confronting cognitions. *Arguing to learn: Confronting cognitions in computer-supported collaborative learning environments*, pages 1–25, 2003.
6. Maurício Aniche, Christoph Treude, and Andy Zaidman. How developers engineer test cases: An observational study. *IEEE Transactions on Software Engineering*, 48(12):4925–4946, 2021.
7. Michael J Baker. Argumentation and constructive interaction. *Foundations of argumentative text processing*, 5:179–202, 1999.
8. Sebastian Baltes and Paul Ralph. Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering*, 27(4):94, 2022.
9. Andrew Begel and Nachiappan Nagappan. Pair programming: What's in it for me? New York, NY, USA, 2008. Association for Computing Machinery.
10. Sandro Bimonte, Kamal Boulil, François Pinet, and Myoung-Ah Kang. Design of complex spatio-multidimensional models with the icsolap uml profile-an implementation in magicdraw. In *International Conference on Enterprise Information Systems*, volume 2, pages 310–315. SCITEPRESS, 2013.
11. Bernd Bruegge and Allen H. Dutoit. *Object-Oriented Software Engineering Using UML, Patterns, and Java.* Prentice Hall Press, 3rd edition, 2009.
12. Lola Burgueño, Antonio Vallecillo, and Martin Gogolla. Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, 28(1):23–41, 2018.
13. Lola Burgueño, Antonio Vallecillo, and Martin Gogolla. Teaching UML and OCL models and their validation to software engineering students: an experience report. *Computer Science Education*, pages 1–19, 04 2018.
14. Jeff Carver, Forrest Shull, and Victor Basili. Observational studies to accelerate process experience in classroom studies: An evaluation. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 72–79. IEEE, 2003.
15. Shalini Chakraborty and Grischa Liebel. We do not understand what it says–studying student perceptions of software modelling. *Empirical Software Engineering*, 28(6):149, 2023.
16. Shalini Chakraborty and Grischa Liebel. Evaluating software modelling recommendations: Towards systematic guidelines for modelling, 2024.
17. Shalini Chakraborty and Grischa Liebel. Modelling guidance in software engineering: a systematic literature review. *Software and Systems Modeling*, 23(1):249–265, 2024.
18. Federico Ciccozzi, Gabi Taentzer, Antonio Vallecillo, Manuel Wimmer, Michalis Famelis, Leen Lambers, Sebastien Mosser, Richard Paige, Alfonso Pierantonio, Arend Rensink, and Rick Salay. How do we teach modelling and model-driven engineering? a survey. In *Proceedings of the 21st ACM/IEEE international conference on model driven engineering languages and systems: Companion proceedings*, pages 122–129, 2018.
19. Uri Dekel and James D Herbsleb. Notation and representation in collaborative object-oriented design: an observational study. *ACM SIGPLAN Notices*, 42(10):261–280, 2007.
20. Andrew Forward, Omar Badreddin, and Timothy 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.

21. Fabian Gilson. Teaching software language engineering and usability through students peer reviews. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 98–105, 2018.
22. Laure Gonnord and Sébastien Mosser. Practicing domain-specific languages: from code to models. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, pages 106–113, 2018.
23. Tony Gorschek, Ewan Tempero, and Lefteris Angelis. On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software*, 95:176–193, 2014.
24. Imed Hammouda, Håkan Burden, Rogardt Heldal, and Michel RV Chaudron. Case tools versus pencil and paper. In *ACM/IEEE 17th Int. Conf. on Model Driven Engineering Languages and Systems–Educators Symposium*, 2014.
25. John Hutchinson, Mark Rouncefield, and Jon Whittle. Model-driven engineering practices in industry. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 633–642, 2011.
26. John Hutchinson, Jon Whittle, Mark Rouncefield, and Steinar Kristoffersen. Empirical assessment of mde in industry. In *2011 33rd International Conference on Software Engineering (ICSE)*, pages 471–480, 2011.
27. Sascha Kirstan and Jens Zimmermann. Evaluating costs and benefits of model-based development of embedded software systems in the car industry–results of a qualitative case study. In *Workshop C2M: EEMDD "From code centric to model centric: Evaluating the effectiveness of MDD"*, 2010.
28. Dimitrios S Kolovos and Jordi Cabot. Towards a corpus of use-cases for model-driven engineering courses. In *EduSymp/OSS4MDE@ MoDELS*, pages 14–18, 2016.
29. Maria Kutar, Carol Britton, and Trevor Barker. A comparison of empirical study and cognitive dimensions analysis in the evaluation of uml diagrams. In *Procs of the 14th Workshop of the Psychology of Programming Interest Group (PPIG 14)*, pages 1–14, 2002.
30. Ludwik Kuzniarz, Miroslaw Staron, and Claes Wohlin. An empirical study on using stereotypes to improve understanding of uml models. In *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.*, pages 14–23. IEEE, 2004.
31. Grischa Liebel. *Model-Based Requirements Engineering in the Automotive Industry: Challenges and Opportunities*. Chalmers Tekniska Hogskola (Sweden), 2016.
32. Grischa Liebel, Rogardt Heldal, and Jan-Philipp Steghöfer. Impact of the use of industrial modelling tools on modelling education. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET)*, pages 18–27, 2016.
33. Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software and Systems Modeling*, 17(1):91–113, 2018.
34. Grischa Liebel, Matthias Tichy, and Eric Knauss. Use, potential, and showstoppers of models in automotive requirements engineering. *Software and Systems Modeling*, 2018.
35. Adriana Lopes, Igor Steinmacher, and Tayana Conte. UML acceptance: Analyzing the students' perception of UML diagrams. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, pages 264–272, 2019.
36. Nicolas Mangano, Thomas D LaToza, Marian Petre, and André van der Hoek. How software designers interact with sketches at the whiteboard. *IEEE Transactions on Software Engineering*, 41(2):135–156, 2014.
37. Naomi Miyake. Constructive interaction and the iterative process of understanding. *Cognitive science*, 10(2):151–177, 1986.
38. Parastoo Mohagheghi, Wasif Gilani, Alin Stefanescu, Miguel A. Fernandez, Bjørn Nordmoen, and Mathias Fritzsche. Where does model-driven engineering help? experiences from three industrial cases. *Software and Systems Modeling*, 12(3):619–639, 2013.
39. Leonel Nóbrega, Nuno Jardim Nunes, and Helder Coelho. The meta sketch editor: a reflexive modeling editor. In *Computer-Aided Design Of User Interfaces V*, pages 201–214. Springer, 2007.

40. Richard F Paige, Fiona AC Polack, Dimitrios S Kolovos, Louis M Rose, Nicholas Drivalos Matragkas, and James R Williams. Bad modelling teaching practices. In *EduSymp@ MoDELS*, pages 1–12, 2014.
41. Tanumoy Pati, Sowmya Kolli, and James H Hill. Proactive modeling: a new model intelligence technique. *Software & Systems Modeling*, 16:499–521, 2017.
42. Jakob Pinggera, Pnina Soffer, Dirk Fahland, Matthias Weidlich, Stefan Zugal, Barbara Weber, HajoA Reijers, and Jan Mendling. Styles in business process modeling: an exploration and a model. *Software and Systems Modeling*, pages 1–26, 05 2013.
43. Jakob Pinggera, Pnina Soffer, Stefan Zugal, Barbara Weber, Matthias Weidlich, Dirk Fahland, Hajo Reijers, and Jan Mendling. Modeling styles in business process modeling. volume 113, 01 2012.
44. Parsa Pourali and Joanne M. Atlee. An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, MODELS '18, page 224–234, New York, NY, USA, 2018. Association for Computing Machinery.
45. Alexander Ragnarsson, Shalini Chakraborty, and Grischa Liebel. Modrec: A tool to support empirical study design for papyrus and the eclipse modeling framework. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 361–369, 2021.
46. Paul Ralph, Sebastian Baltes, Domenico Bianculli, Yvonne Dittrich, Michael Felderer, Robert Feldt, Antonio Filieri, Carlo Alberto Furia, Daniel Graziotin, Pinjia He, Rashina Hoda, Natalia Juristo, Barbara A. Kitchenham, Romain Robbes, Daniel Méndez, Jefferson Molleri, Diomidis Spinellis, Miroslaw Staron, Klaas-Jan Stol, Damian A. Tamburri, Marco Torchiano, Christoph Treude, Burak Turhan, and Sira Vegas. ACM SIGSOFT empirical standards. *CoRR*, abs/2010.03525, 2020.
47. Rebecca Reuter, Theresa Stark, Yvonne Sedelmaier, Dieter Landes, Jürgen Mottok, and Christian Wolff. Insights in students' problems during UML modeling. In *2020 IEEE Global engineering education conference (EDUCON)*, pages 592–600. IEEE, 2020.
48. Rebecca Reuter, Theresa Stark, Yvonne Sedelmaier, Dieter Landes, Jürgen Mottok, and Christian Wolff. Insights in students' problems during UML modeling. In *2020 IEEE Global Engineering Education Conference (EDUCON)*, pages 592–600, 2020.
49. Jason E. Robbins and David F. Redmiles. Cognitive support, uml adherence, and xmi interchange in argo/uml. *Information and Software Technology*, 42(2):79–89, 2000.
50. James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William E. Lorensen, et al. *Object-oriented modeling and design*, volume 199. Prentice-hall Englewood Cliffs, NJ, 1991.
51. Andreas Schmidt, Daniel Kimmig, Klaus Bittner, and Markus Dickerhof. Teaching model-driven software development: Revealing the" great miracle" of code generation to students. In *Proceedings of the Sixteenth Australasian Computing Education Conference-Volume 148*, pages 97–104, 2014.
52. Dave R. Stikkolorum, Truong Ho-Quang, and Michel 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*, pages 275–279, 2015.
53. Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. Relevance, benefits, and problems of software modelling and model driven techniques—a survey in the italian industry. *Journal of Systems and Software*, 86(8):2110–2126, 2013.
54. Arja L Veerman, JEB Andriessen, and Gellof Kanselaar. Collaborative learning through computer-mediated argumentation. 1999.
55. Bernd Westphal. Teaching software modelling in an undergraduate introduction to software engineering. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 690–699, 2019.
56. Jon Whittle and John Hutchinson. Mismatches between industry practice and teaching of model-driven software development. In *International Conference on Model Driven Engineering Languages and Systems*, pages 40–47. Springer, 2011.

57. Jon Whittle, John Hutchinson, and Mark Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, 2014.
58. Jon Whittle, John Hutchinson, Mark Rouncefield, Håkan Burden, and Rogardt Heldal. Industrial adoption of model-driven engineering: Are the tools really the problem? In *International Conference on Model Driven Engineering Languages and Systems*, pages 1–17. Springer, 2013.
59. Laurie Williams and Robert R Kessler. *Pair programming illuminated*. Addison-Wesley Professional, 2003.
60. Laurie Williams, Charlie McDowell, Nachiappan Nagappan, Julian Fernald, and Linda Werner. Building pair programming knowledge through a family of experiments. In *2003 International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 143–152. IEEE, 2003.
61. Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering*. Springer Science & Business Media, 2012.
62. Dina Zayan, Michał Antkiewicz, and Krzysztof Czarnecki. Effects of using examples on structural model comprehension: a controlled experiment. In *Proceedings of the 36th International Conference on Software Engineering*, pages 955–966, 2014.

A Problem Descriptions

We present three problem descriptions used throughout our study setup. We changed the details and timings after two pilots. The main study was for 1 hour, divided into Reading the full problem description (5 minutes), one of the participants solving task 1 with a class diagram (25 minutes), taking a small break (5 minutes), the other participant solving task 2 with sequence diagrams (25 minutes).

A.1 Pilot Study 1

- **Problem Description**: The task is to model the domain of a restaurant food court system using class diagram. The system should facilitate the management of multiple restaurants within a food court, customer orders, and the preparation and delivery of food. Your model should accurately represent the relationships between the entities involved and provide a clear structure for the system's operations. Please consider the following requirements for the system: The system must support multiple restaurants within the food court, each restaurant should have details such as name, cuisine type, menu items, and operating hours, an order should include customer details, a list of ordered items, total price, and order status (e.g., pending, preparing, completed) and finally, the system should support the delivery of orders to customers, including delivery time and delivery personnel details.
- **Modelling Tool**: Please use the Eclipse Papyrus for the task.
- **Time**: 30 minutes

A.2 Pilot Study 2

- **Problem Description** The task involves modelling two problems related to an international dating app. Some information of the system: to create a profile, users must provide their personal information, location, and at least three pictures. Every user should have a valid credentials. The app operates by allowing users to see others within a certain geographic area, defined as a flexible distance (x km) from the user's location, which can be either their current GPS location or a manually set coordinate. To initiate a connection, a user can send a "like" to another user within the same geographic area.

The recipient will receive a notification with the sender's profile information and has two days to confirm the like. If there is no confirmation within two days, the like is deleted.
The first task is to model the domain using class diagram. The second task is to model two behaviours of the system: user profile creation and initiating connection between two users.
- **Modelling Tool**: Please use a modelling tool of your choice.
- **Switching Roles**: After a brief description of pair programming, students were instructed here to switch roles between tasks.
- **Time Limit**: Total 1 hour. Read the problem: 10 minutes, class Diagram: 20 minutes, small break: 10 minutes and sequence Diagram: 20 minutes.

A.3 Main Study

- **Problem Description**:Your job here is to solve two tasks regarding a dating app. You must maintain the following constraints while solving the tasks:
  1. It is an international dating app, not country specific.
  2. To open a profile, the user must provide their name, email id, gender preference, location and at least 3 pictures. Every user should have a username and password.
  3. How does this app work? Assume User1 and User2 are both users of this app from specific geographic areas. Two things to notice here: "geographic area" and "user's location". A "geographic area" is a certain distance (x km) from the user's location, which is flexible. A "user's location" can be either that user's current GPS location or some other coordinate set by the user manually. For example, if User1's location is Reykjavik, User1 can set his/her/their geographic location within 80 km of Reykjavik.
  4. User1 can see all other users of this app with the same geographic location.
  5. To initiate a connection an interested user of this app send a "like". User1 can like User2 if both share the same geographic location. User2 will get a notification, including User1's profile information. User2, in this case, has two days to confirm User1's like; a confirmation means User2 also like User1. If there is no confirmation, after two days User1's like will be deleted.
  6. If both User1 and User2 send like to each other, the app initiates a message thread private to both users. If none of the users starts talking after 1 week, the thread will be deleted, including both users' like.
- **Task 1: Class Diagram** Your task is to model the domain of this app, that is, design the domain model using a class diagram. Please mention the attributes of each class and the multiplicities. Try to think about the operations for each class and mention as many as you can. The purpose here is for a pair to decide what should be in the system and what's outside of the system.
- **Task 2: Sequence Diagram** Your task is to model the sequence of the following functionalities:
  1. User profile creation: Show step-by-step how a user creates a profile in the app. Try to be as detailed as possible.
  2. Users initiate the connection: Now that users have created their profiles, show step-by-step how a connection is made between two users. (Use the description given above to understand the communication between the system and its users, to draw the sequence diagram)

- **Modelling Tool**: Please use a modelling tool of your choice.
- **Switching Roles**: After a brief description of pair programming, students were instructed here to switch roles between tasks.
- **Time Limit**: Total 1 hour. Read the problem: 5 minutes, class Diagram: 25 minutes, small break: 5 minutes and sequence Diagram: 20 minutes.

Chapter 9

Paper IV: Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling

Shalini Chakraborty, Grischa Liebel
18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (EMSE'24)
https://doi.org/10.1145/3674805.3686693

Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling

Shalini Chakraborty
shalini19@ru.is
Reykjavik University
Reykjavik, Iceland

Grischa Liebel
grischal@ru.is
Reykjavik University
Reykjavik, Iceland

Abstract

Background: Despite having several advantages, software modelling remains unpopular for developers. Similarly, university students do not see the benefits of software modelling in the university curriculum. Prior research show the lack of guidance for students to do so.

Aims: We aim to evaluate the effectiveness of four modelling recommendations made in related work to improve student modelling knowledge. Additionally, we aim to discover students' perceptions of software modelling after taking a course with the recommendations included.

Method: We conducted a mixed method study, including interviews with teaching assistants, student surveys, and a focus group study involving students, teaching assistants, and experts from both modelling and education.

Results: We find that the four recommendations overall have a positive impact as they help students better understand the modelling knowledge from the course. Students express that specific recommendations help them grasp the concept of software modelling well. We also extend the recommendations by adding more details specific to software modelling and solidifying the recommendations into systematic guidelines.

Conclusions: The guidelines can potentially enhance education and training in software modelling, catering to both academic settings and industrial environments. Additionally, the guidelines contribute to improved communication between students and the course itself by outlining what students can expect from modelling assignments and the value inherent in each of these assignments.

CCS Concepts

• **Software and its engineering → Unified Modeling Language (UML)**.

Keywords

Software Modelling, UML, Education, Interview, Survey, Focus Group

ACM Reference Format:
Shalini Chakraborty and Grischa Liebel. 2024. Evaluating Software Modelling Recommendations: Towards Systematic Guidelines for Modelling .

ESEM '24, October 24–25, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-1047-6/24/10
https://doi.org/10.1145/3674805.3686693

In *Proceedings of the 18th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), October 24–25, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3674805.3686693

1 Introduction

Software modelling has a significant influence on software development and product maintenance. Despite having multiple benefits [2, 15], software modelling remains unpopular among developers [12]. The picture is similar in education. Often while learning modelling, university students limit their modelling application to course assignments. Students recognise specific advantages of modelling, particularly in areas where informal models suffice, such as understanding a system overview or conceptual comprehension of the problem domain. However, most students still need to be convinced about the value of modelling in detailed system design. Many students expressed challenges related to issues such as selecting the appropriate notation, determining what aspects to express in the models, and understanding how to apply modelling techniques to unfamiliar domains [5, 20, 30].

In our previous case study [5], we present four recommendations for modelling education in universities after an extensive case study based on three universities. The recommendations are to be applied to modelling courses or courses where modelling is a sub-part. The recommendations included assignment structures, grading instructions and feedback suggestions. In this study, we aim to evaluate the four recommendations and examine students' perception of modelling after taking a course with the recommendations.

Therefore, to address this goal we have the following research questions (RQs):

- RQ1: How effective are the recommendations in improving students' software modelling skills?
- RQ2: What are students' perceptions of software modelling after the implementation of the recommendations in the courses?

We conducted a mixed-method empirical study to answer the RQs. To explore the effectiveness of the recommendations, we applied them in two 12-week bachelor courses related to software design. Afterwards, we continued the empirical study with a qualitative investigation. This involves gathering feedback from students (through surveys) and instructors (through interviews) regarding the impact of the applied modelling recommendations. To delve into students' actual understanding of modelling post-recommendation implementation, we analysed student assignments in addition to their feedback. This analysis focused on assessing the syntax and semantics of UML diagrams created as a part of their assignments, aiming to gain insights into students' understanding of modelling

concepts imparted through the courses. Finally, we conducted a focus group study with students, teaching assistants, and experts from both modelling and education to get early feedback on the overall impact of our recommendations.

Our findings show that the recommendations have a positive impact on modelling education. Students and teaching assistants expressed that specific recommendations helped them communicate better with each other and the course. Students expressed that the recommendations helped them become aware of the assignments' expectations, which allowed them to distribute effort into the assignment and take the knowledge of software modelling at their own pace. Although we observed a few challenges, such as a lack of semantic understanding of the modelling diagrams, students could grasp the modelling knowledge through familiar domains. Apart from the initial evaluation, we added specific details to the recommendations to solidify them into systematic guidelines, which will help students in the long run and better carry the knowledge from university into industry careers.

The rest of the paper is structured as follows: we discuss the related work in Section 2, Section 3 demonstrates our methodology and data analysis strategies. We report the results of our study in Section 4, followed by discussions in Section 5 conclusion in Section 6.

2 Related Work

In the following, we describe related work that discusses research on modelling education, especially how to teach modelling and students' role in empirical studies.

2.1 Research on Modelling Education

Modelling and model-based software development represent crucial components of software engineering [6, 33]. There exists a substantial number of literature describing the inclusion of modelling and modelling languages like UML into general software engineering or software design courses [7, 10, 18, 25, 32, 33].

Rapos [25] describes a course based on model-driven software engineering (MDSE) with a higher emphasis on lab performance and assignments based on students' feedback. Gonnord et al. [11] guide students from low-level C code to crafting a modelling language workbench. Westphal [33] outlines the structure of an SE course heavily centred on modelling. Gilson et al.[9], advocate for a course where students serve both as language designers and users to assess the usability of software language engineering (SLE). Vallino [32] propose and describe a course where they claim that should be emphasized throughout the curriculum. In each instance, student feedback forms the cornerstone of evaluating the proposed course design.

Several empirical studies have discussed modelling and especially UML in education [1, 5, 13, 21, 26, 30]. Agner et al. [1] conducted a survey on the utilisation of modelling tools among 117 students. Their findings highlighted issues such as inadequate feedback, challenges in diagram creation, and tool complexity. Hammouda et al. [13] compared the usage of modelling tools to traditional pen-and-paper methods. Through a survey, they assessed students' perceptions of the differences, ultimately finding no distinct advantage for either approach. Reuter et al. [26] investigated students' challenges with UML diagrams across two modelling courses. In a similar vein, Stikkolorum et al. [30] explored student difficulties and modelling approaches in UML Class diagrams by providing students with a specialised UML editor equipped with feedback mechanisms.

2.2 Students' Role in Empirical Studies

Empirical studies with students, especially in the are of software modelling is not uncommon. Pourali et al. [24] describes a two-phase user study aimed at identifying the primary challenges users encounter while developing UML Class and State-Machine diagrams using UML modelling tools. Badreddin et al. [4] explore the impact of education on students' perceptions of modelling by conducting a survey across three countries. Kuzniarz et al. [19] examine the use of stereotypes examples to enhance the understanding of UML models. Kutar et al. [17] perform an empirical study to assess whether users' comprehension of information in Sequence diagrams differs from that in Collaboration diagrams. Chakraborty and Liebel [5] conduct a multiple case study to explore students' perceptions of software modelling. The study involve conducting 21 interviews with students and instructors across five courses at three universities.

3 Method

In this empirical study, we engaged with university students to evaluate the efficacy of four modelling recommendations introduced during their bachelor courses.

3.1 Recommendations

In our previous case study [5], we engaged in 16 interviews with students and conducted 5 interviews with instructors from three different universities to explore students' perceived understanding of modelling and the challenges they encountered.

After studying the benefits and challenges of students, we present four recommendations for modelling education. Below, we discuss the four recommendations and our explanations behind them.

R1 Iterative Projects with Regular Feedback: We recommend iterative assignments/projects to help students practice what they have learned. Along with that, instructors must provide regular feedback. Regular feedback is consistent, timely, specific and contains assessment details of every assignment students were instructed to do. Combining iterative projects and regular feedback can make learning more efficient as students get to implement the feedback during the course time.

R2 Limitation of diagram types: UML constitutes a vast and intricate domain, presenting a substantial challenge for learners. To enhance the learning experience, we advocate for a more realistic approach that acknowledges the limitations of students' time and cognitive capacity. Hence, we propose narrowing down the array of diagram types covered in a course. By being more realistic about what students can feasibly learn within a limited time frame, we aim to prevent overwhelming them with an extensive variety of diagrams. In our study[5], we encounter the challenge of "unclear expectations," where students struggle to align a given problem with the appropriate modelling diagram. We believe that reducing the variety of diagrams becomes crucial to address that challenge,

allowing students to delve deeper into the intricacies of each diagram type and fostering a clearer understanding of their respective purposes.

R3 Grading Rubrics: Along with *irregular feedback*, we observe confusion regarding assessment. Irregular feedback refers to feedback that is provided inconsistently. In our case study, we noticed that students received feedback at varying times rather than in a systematic or scheduled manner. This lead to confusion, uncertainty, and difficulty in understanding how to improve or progress, as they may not know when to expect feedback or may receive it too late to make effective changes. Unlike coding, the instructors must provide students with constructive and regular feedback, as modelling doesn't offer runtime errors as programming. At least not in the early stages when students are not generating code from the models. Therefore, we recommend having grading rubrics for assignments where stated model purposes and grading are aligned, and aligning that with each task is very important for the student's understanding. In the rubric, assignments are broken down into tasks, and priorities and grades for every task are mentioned. Grading rubrics foster better communication between students and instructors, allowing instructors to provide constructive feedback regularly. A sample rubric is attached as a supplementary file[1]

R4 Use of Familiar Problem Domains: Finally, we discussed the challenge of the *absence of expertise* in the specific problem domain. To overcome this hurdle, we recommend educators consider employing familiar problem domains, perhaps ones selected by the students.

3.2 Study Design

Empirical evidence holds significant importance in evaluating various aspects as it provides tangible data and observations derived from real-world experiences or experiments. We selected qualitative analysis of the empirical evidence [28]. Qualitative research methods were primarily developed by educational researchers and other social scientists to delve into the intricacies of human behaviour, focusing more on words than numbers [8, 22]. In our context, qualitative analysis was apt as we were assessing recommendations intended for human application in the creative task of modelling. We incorporated the recommendations in two university courses, focusing on software design and emphasising UML modelling. One of the authors assumed the role of instructor in both instances. We integrated a combination of the suggested recommendations to optimise the learning experience. Both courses were part of the same university and designed for bachelor-level students. For the data collection process, we gathered students' assignments along with their questions throughout the assignment phases. Post-course, we conducted a survey to gain insights into their overall experience. We interviewed teaching assistants (TAs) for their perspectives on the recommendations and overall student participation. We used informed consent to gather all data. Students signed a consent form, consenting to grant access to their assignments and grading information for this study. The post course survey was voluntary for students. We also obtained informed consents from TAs for the interviews. In both surveys and interviews, participants had the right to withdraw at any moment, and they also retained the right to withdraw their consent at any time.

In the next sections, we explain the courses in more detail and show how we used the recommendations in our teaching approach.

C1: Software Analysis and Design The course is given to first-year bachelor students in Computer Science. The curriculum broadly addresses aspects like software prototype design, user interfaces, requirement analysis, and UML modelling. Specifically, the first author took charge of the UML section, covering 30 percent of the entire course. We crafted the entire syllabus for UML, including assignments and grading criteria. In total, the course accommodated 259 students, out of which 50 students granted permission for us to use their assignments via signed consent forms. Additionally, 30 students out of 259 provided feedback through a post-course survey. The survey was anonymous.

We incorporated R1, R2, and R3 in the course design. For R1, we had trouble providing students with regular feedback because it was an extensive course, and we had six different TAs (teaching assistants). As the TAs are responsible for grading assignments and giving feedback, we observed a few communication issues.

We did not select the problem domain for this course, as another instructor started the course with modelling coming later in the course. We aimed for continuity in student engagement with the initial problem domain (following R1), so we adhered to the same domain throughout and tailored assignments accordingly. The domain was Icelandic tourism, and students were working to make software that provides tourists with information about tourist places in Iceland, their availability, weather forecasts and facilities around these places for efficient visits.

C2: System Analysis and System Design The course is given to first-year bachelor students in Software Engineering, placing a heightened emphasis on modelling compared to the previous course. The syllabus encompasses a range of topics, including requirement engineering, diverse interface design methodologies, UML modelling covering 6 types of UML diagrams (use case, class, sequence, activity, state, and component diagrams) and their practical application, and testing of the designs. The first author was in charge of the complete course. Of the 37 enrolled students, 35 granted us permission to use their assignments via consent forms and 31 students contributed to the post-course anonymous survey.

We incorporated all four recommendations in the course design. As a problem domain, we surveyed pre-course to know what students would like to work on and finalised four options from their answers:

StudyBuddyConnect, a free online platform exclusively for university students, where they can enrol, register qualifications, and find study partners.

IcelandThrillQuest, a platform for discovering thrilling physical activities in Iceland.

MoodMusic a system to define tracks one loves and emotions one feels when listening.

DiverseConnection, a nonprofit digital story-sharing platform fostering inclusivity and creativity where individuals of all genders, races, sexes and backgrounds can unite to share their passion for art, music, and dance.

We opted for both of these courses due to the shared instructor, with the same author teaching both. Additionally, both courses

[1] https://doi.org/10.5281/zenodo.11125637

feature modelling in their syllabus, providing a convenient opportunity for us to implement the recommendations. The timing of these courses further worked to our advantage, given that they were scheduled consecutively—C1 in the fall semester (2023) and C2 in the spring (2024).

Table 1 describes the detailed actions we took to address the four recommendations.

In addition to the mentioned actions, we granted students the freedom to select their preferred tools. We firmly believe that this freedom carries its benefits, increasing the likelihood of students encountering a positive initial modelling experience rather than becoming frustrated with tool-related challenges.

3.3 Data Collection and Data Analysis

We collected three types of data from both courses[2]: *student feedback*, *instructor's feedback regarding grading the assignments*, and finally, the *assignments performed by the students*.

Student feedback: For the first data source, *Student feedback*, we aimed to understand how students perceive the newly designed UML lectures and assignments. We surveyed at the end of the course. For C1, the survey was voluntary, and we received 30 survey participants. For C2, we made the survey the last assignment of the course, carrying 5 per cent of the final grade. We received 31 responses. The survey included questions about their learning (challenges and benefits), given materials, applied recommendations, specific diagrams, etc. For C2, we kept the same questions as C1, just added two more questions as we applied all 4 recommendations in C2[3].

Instructor's feedback: For the second data source, *instructor's feedback*, we interviewed teaching assistants (TAs) from both courses[4]. TAs are responsible for interacting with students during the problem-solving sessions[5] and grading the assignments. We interviewed 4 TAs from C1. Of the four TAs, two were involved with the course in the previous academic year (fall semester, 2022), and thus, we asked them a few additional questions to get their opinion for both academic years. We aimed to compare both years and evaluate the effectiveness of our modelling recommendations.

For C2, we had one TA. This person was a TA in C1 as well. We had a slightly different interview guide for C2 due to the different nature and types of assignments.

The instructors are identified as TA_number_course, for example: TA_1_C1, TA_2_C1, TA_3_C1, TA_4_C1 and TA_C2.

Assignments: We structured the assignments to allow students to practice modelling twice. Initially, in both courses, students completed small assignments (typically carrying 2 percent and 5 percent of the overall grade) to draw the diagrams for the first time. Subsequently, we introduced group assignments where students could read diagrams from their peers, share ideas within their groups, and then redraw the diagrams. This collaborative approach helped students gain a deeper understanding of the problem domain and enhanced their diagramming skills.

We gathered the small assignments and group assignments related to UML diagrams from students. In total, we compiled 3 small assignments and 1 group assignment from C1 (from 50 students), as well as 4 small assignments and 2 group assignments from C2 (from 35 students).

Additionally, we conducted a focus group [16] session to evaluate the recommendations. Prior to recording the focus group study, we obtained consent from all participants. Moreover, students received a movie voucher for their participation in the focus group study. The recommendations mentioned in [5] are in the early stages of development. Therefore, employing focus groups is helpful, as they provide an excellent opportunity to gather initial feedback. The specific objective is to evaluate the recommendations within a defined context and explore participants' opinions and experiences regarding their use.

- **Selecting Participants**: We chose 6 participants for the focus group, representing various aspects and experience levels. This included 2 students, one TA (who was a TA in both C1 and C2), one presenter and host of the focus group, one expert in software modelling, and finally, one representative from the education sector.
- **Planning and Conducting the Session**: The session was planned for 1 hour. The host started the session by presenting the research and recommendations. Each participant then presented their opinions on them. The host also acted as a facilitator. Their role during the sessions involved:
  (1) Ensuring that crucial topics were addressed.
  (2) Keeping discussions on track.
  (3) Prompting participants to share their viewpoints.
  However, the facilitator refrained from actively participating in the discussion. Alongside facilitating, they were responsible for taking notes during the discussions. The discussion was semi-structured. There were specific themes. However, the themes helped direct the conversation without constraining participants from having an open discussion.

For data analysis, we chose three approaches. Firstly, we conducted in-vivo coding to analyze the feedback received from both students and instructors. Secondly, for the assignments, we developed an analysis protocol: Diagram Assessment Sheet or DAS. Following this protocol, we thoroughly examined all the assignment submissions, which included diagrams drawn by the students. Finally, for the focus group, both researchers opted for open code analysis.

In-vivo Coding: In-vivo coding, as described in [27], is utilized as one of the initial cycle coding methods for qualitative analysis. It involves directly using the literal spoken words as codes, rather than creating one's own terms (open coding).

Initially, we transcribed all the interview data from C1, which included interviews with the four TAs. Subsequently, both researchers independently coded one interview each, followed by a discussion of the outcomes. Once we reached an agreement, we proceeded to code the remaining interviews. We repeated the same process for

[2]We asked for students' consent in the beginning and collected data only for those students who agreed to our data collection. Participation was voluntary, and participants may withdraw from the study at any time, without penalty. Furthermore, participants could deny the researchers conducting the study the right to use the collected data (from their participation) and the right to publish the anonymised transcripts.
[3]Survey questions are added to: https://doi.org/10.5281/zenodo.11125637
[4]The interview guide and transcripts are added to: https://doi.org/10.5281/zenodo.11125637
[5]Students perform the assignments in problem-solving sessions. In our case, students solve the individual assignments in problem-solving sessions and work on the group assignments.

Recommendations	Actions taken in Courses
R1	The course project was divided into two forms: individual and group assignments. Assignments were designed to allow students to apply what they learned in individual tasks to group projects. This approach enables students to analyse and implement the feedback received on individual assignments in their group work.
R2	We narrowed down the syllabus to focus on a few types of diagrams. Due to time constraints in C1, we concentrated on just three diagram types and created assignments based on them. In C2, we extended the learning period to four weeks, covering six diagram types. Students were given the opportunity to complete four individual assignments and two group assignments related to these six diagram types (In C1, students performed 3 individual assignment and 2 group assignment). In both courses, individual assignments were used for students to initially learn and practice a specific type of diagram. In the group assignments, students were required to model the diagrams they had learned in the individual assignments, but this time collaboratively. This approach allowed students to apply the feedback they received on a particular diagram, work with peers, and engage in collaborative modelling. The problem domain remained consistent across all assignments.
R3	Grading rubrics were employed for both individual and group assignments. In both courses, the instructor introduced grading rubrics in the first lecture and explained how they would work throughout the course. The grading system was structured to align with specific tasks, ensuring that students comprehended the significance of each step in the assignments. Feedback corresponded to the rubric, facilitating easy and systematic analysis for students. We published rubrics at the same time as the assignments. Hence, students knew the evaluation criteria before modelling.
R4	In C1, we couldn't incorporate R4. Nevertheless, we made sure that all modelling assignments revolved around the same problem students had been working on since the start. In C2, we encouraged them to propose their own domains. This approach aimed to actively involve students in the process and enhance their sense of inclusion. We also ensured the students worked on the same domain for the entirety of the course to increase their understanding and comfort.

Table 1: Various actions taken in both courses (C1 and C2) based on four recommendations (R1, R2, R3 and R4)

all 30 student surveys for C1 and the interviews and surveys from C2.

Diagram Assessment Sheet or DAS: We developed an assessment strategy called the Diagram Assessment Sheet (DAS) to assess students' understanding of UML diagrams in the course based on their created models/UML diagrams. DAS evaluates two key aspects of a UML diagram: Syntax and Semantics. It's important to note that the DAS is aligned with the proposed recommendations in [5] and the way modelling was taught in those courses, as outlined in Section 3.1. Therefore, we are assessing students' comprehension of modelling within the framework of the recommendations. Our goal is to evaluate the students' overall knowledge of software modelling based on the recommendations.

The assessment sheet includes students' data for each of their modelling assignments. Each assignment is accompanied by syntax and semantics columns, where the syntax column employs a numbering system to denote incorrect notations with a (-1) marker. In contrast, the semantics column adopts an open coding approach, documenting all semantic issues encountered in the diagrams. Subsequently, both authors employed DAS to analyse two students' assignments individually. Following this initial analysis, the authors discussed their findings, refined DAS and conducted further analysis on an additional ten students' assignments. Once a consensus was reached on the analysis strategies within DAS, the analysis was extended to encompass all 50 students' assignments from C1 and 35 students from C2.

Focus Group Analysis: We recorded the focus group session, transcribed the data and put them into an Excel sheet[6]. We then separated research explanations, clarification points and other informal/non-related communications from the data. We categorised the rest of the communication into three different aspects: benefits of the recommendations, challenges of the recommendations and modifications of the recommendations.

3.4 Validity Threats

We conducted a mixed-method empirical study, primarily collecting data through interviews, surveys and a focus group study. Afterwards, we took an interpretive approach [14, 23] and analysed the data via qualitative measures. An interpretivist approach implies that humans construct knowledge as they interpret their experiences of and within the world. We opted for interpretivism to evaluate the recommendations as it was essential to understand our study participants' experience with modelling recommendations and their understanding of modelling afterwards. A potential threat to our study is that the four recommendations were applied in parts of two courses. As the courses have more to offer, the other parts (such as non-modelling theory/assignments) influence students' experience of the recommendations, which leads to the perception of modelling. To address this issue, we gathered students' opinions and real experiences through surveys and assignments. We also

[6]https://doi.org/10.5281/zenodo.11125637

supplemented the data with the perspectives of teachers by interviewing TAs. Additionally, we held a focus group study to obtain more targeted feedback on the recommendations from students, TAs, and experts in modelling and education. We acknowledge that our data set consists of just two university courses. We tried to mitigate that by selecting two courses from two bachelor fields (Computer science and software engineering).

To ensure the credibility of our findings, we based our analysis on in-vivo codes taken directly from the exact words people said in the interviews. Both authors did the coding. We also had a one-hour focus group discussion to hear different opinions about recommendations, where we included experts in education and modelling who were not part of any of the two courses. Moreover, in a commitment to transparency, we have made anonymised transcripts available for interviewees who granted consent.

One of the authors served as the modelling instructor in both courses. There is a potential for researcher bias when teaching modelling through the recommendations. However, we address this concern by involving TAs in grading assignments. Additionally, TAs were responsible for practical sessions. For instance, R3 (grading rubrics) were employed by students and TAs without any input from the instructor.

In C1, we had over 280 students and seven TAs grading the entire course. The modelling segment occurred towards the end of the 12-week course. With previous assignments accumulating, consistent regular feedback for students was challenging to provide. However, students received consistent feedback for the initial two assignments. When they finally received feedback for the last two assignments, they genuinely appreciated its positive impact. In their post-course survey, they emphasised the importance of regular feedback and how it aided their learning.

4 Results

In the following subsections, we present the answers of two RQs with the results of study.

4.1 RQ1: How effective are the recommendations in improving students' software modelling skills?

Our study indicates that the four recommendations are successful in improving students' overall modelling skill. The recommendations are practical as students showed more engagement in assignments, and the communication was significantly better between students and instructors. In the following, we present each recommendation and the evaluation of those based on the study results.

4.1.1 R1 Iterative Projects with Regular Feedback: We observed positive feedback for R1. All four TAs expressed that students were struggling with syntax. Since this was the students' first attempt at drawing UML diagrams, we anticipated some initial difficulty with syntax. However, thanks to R1, students were able to redraw the diagrams using the same problem domain.

> *"I think a lot of them were handing in diagrams that weren't following the syntax. So I think they struggle with understanding the syntax or using the syntax in the first place. They were sometimes using the syntax wrongly, but insisting on using it nonetheless. It got better in group assignments, at least they were asking more specific question with their mistakes."* — TA_1_C1

Upon finishing the course, students came to appreciate learning the syntax. When asked, *"What aspects of the UML modelling part did you find most beneficial to your learning?"* . The majority of students responded learning the syntax was the most valuable aspect.

> *"The most beneficial aspect to learning it was certainly its logical and strict syntax."* — Student_C1

> *"I think the syntax and repeat them again in group assignments. I understood better after that. "* — Student_C1

We asked students, *On a scale of 1 to 5 (where 5 is the highest), how helpful was it to repeat the diagrams in the assignments?*. 60 percent of students rated 5, 30 percent of students rated 4 (see Figure 1) in C1. In C2, 51 percent rated 5 and 19 percent rated 4 (see Figure 2)

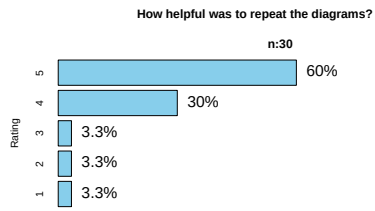


Figure 1: Rating from students of C1: how helpful was it to repeat the diagrams in the assignments

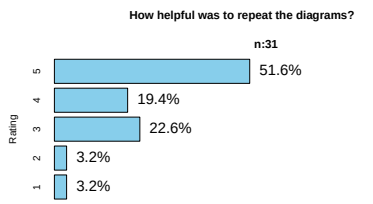


Figure 2: Rating from students of C2: how helpful was it to repeat the diagrams in the assignments

4.1.2 R2 Limitation of diagram types: In C1, due to limited time, we could only include three types of UML diagrams. In C2, we included six types of UML diagrams. We asked students, *On a scale of 1 to 5 (where 5 is the highest), how satisfactory do you think the distribution of UML modelling instruction was?* 45 percent students rated 5 and 41 percent students rated 4 (see Figure 3).

We also asked students of C2, *How do you feel about the time given to learn six diagrams: was it enough for you to understand them well, or do you feel like you needed more time?* Most students mentioned that the time distribution regarding the amount of modelling syllabus was enough for them to get a basic understanding.

> *"I believe that I have acquired a basic understanding of the diagrams that I will build upon in the future through experience."* — Student_C2

> *"But it was enough to get a feeling for each one, so that the next time I see one I at least know what I'm looking at and know what it's telling me."* — Student_C2

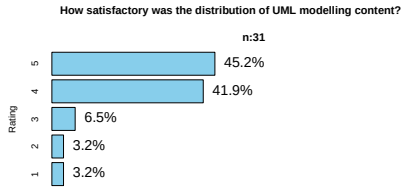


Figure 3: Rating from students of C2: how satisfactory do you think the distribution of UML modelling instruction was

4.1.3 R3 Grading Rubrics. To assess R3, we asked questions to both TAs and students regarding the rubric provided with each assignment. The responses from TAs were overwhelmingly positive, indicating that the rubric aided their comprehension of the grading criteria.

> *"But the rubric was clear. Yeah, okay. I remember it was like, you know, if this is missing the textual mark, if this is missing the text. Yeah, it was very clear in the UML."* — TA_2_C1

We also noticed that the rubric facilitated communication among the TAs. In C1, the rubric helped the six TAs align their grading expectations and ensure consistency.

> *"The rubric made it faster and those meetings [grading meetings among TAs] it was easier to discuss disagreements referring to the instructions"* — TA_4_C1

As modelling involves a level of subjectivity, the rubric may not always offer complete information. Nonetheless, it served as a fundamental guide for TAs to facilitate fair grading.

> *"But then again, like. These types of diagrams have an artistic aspect to them, so it's hard to define a universal rubric. But in general, yeah, like the rubrics had the, the main components and which. Had a direct relation with the assignment description. Please do five of these things. And then like the rubric said one point for each thing four points would be five points. Boom. You can't get any more simple than that. But any ambiguity there is only due to the students' work, not the rubric itself"* — TA_3_C1

For students, the rubric clarified the expectations for each assignment and provided a level of detail that assisted them in prioritising tasks. Additionally, the rubric facilitated feedback, enabling students to formulate questions about its contents. We asked students, *On a scale of 1 to 5 (where 5 is the highest), how helpful were the assignment and grading instructions?* In C1, 63 percent and in C2, 55 percent of students rated 5(see Figure 4 and Figure 5).

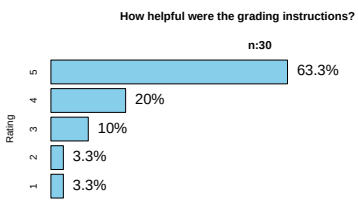


Figure 4: Rating from students of C1: how helpful were the assignment and grading instructions

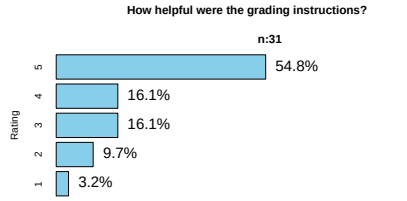


Figure 5: Rating from students of C2: how helpful were the assignment and grading instructions

However, the rubric was not entirely effective when it comes to students assessing their own modelling skills. In the focus group, a student mentioned,

> *"I think it's good for the assignment, but I don't know if you can, like, apply it elsewhere. I think if we don't have the grading rubric, we would probably go a lot more into detail, because we're just doing what's on the grading rubric"* — Student in the focus group

We understood that the rubric served the technical aspects of modelling, which was needed as students were new to modelling and required a mechanical understanding of each UML diagram. However, we felt there is a need to upgrade the rubric so students can use it further to assess their overall modelling skills after the course, along with realistic expectations, which are not limited to modelling assignments but modelling in general.

4.1.4 R4 Use of Familiar Problem Domains. Our findings show that students prefer working with the same problem domain throughout their course. When asked how helpful working with the same problem domain was for all the assignments, students answered positively.

> *"it allowed us to go through the back end development process from gathering requirements to implementing the requirements. It gave us a inside into how the process can be for us in the future"* — Student_C2

Students grew familiar with the domain as the course progressed.

> *"Always when I read the assignment description, I thought to myself how I could implement it with my problem domain."* — Student_C2

Working with the same problem domain help students focus more on UML.

> *"Using the same problem domain highlights the differences between diagrams and such."* — Student_C1
>
> *"That way we could focus better on the actual UML rather than the problem domain"* — Student_C2

4.2 RQ2: What are students' perceptions of software modelling after the implementation of the recommendations in the courses?

We asked students about their overall modelling experience after the course and analysed their assignments to present students' perception of modelling.

4.2.1 Benefits of Modelling. We asked students, *What aspects of the UML modelling part did you find most beneficial to your learning?* Most students mentioned that they found the modelling process and how it can be used in planning and designing software to be most helpful for learning.

> *"Different ways to see a system, helps in planning"* — Student_C1
>
> *"I think all of it. Learning how systems are modelled and designed for the back end"* — Student_C2

Additionally, students found the visual appeal of modelling beneficial for their learning.

> *"Besides understanding how helpful it is for future projects as a student and/or software engineer, I think it also helped me visualise software better and understand how it communicates with itself [different classes] in the big picture"* — Student_C2
>
> *"It was the different ways you build a system and the visual aspect of it"* — Student_C2

Students find that different diagrams provide them with different aspects of software development.

> *"I found learning how to create diagrams for user experience be very beneficial to understanding how software works with its users"* — Student_C1
>
> *"It was fun to connect programming into the diagrams like the class diagram. It was a cool way to see how you could plan your program ahead of time and make a cool diagram so that other people can understand your ideas"* — Student_C1
>
> *"One of the most important UML modelling that I studied in my opinion is the Use case diagram, it gives us an idea on how the system should work which is beneficial when designing a software"* — Student_C2

Further, we asked students, *which specific topics or diagrams within UML did you find most interesting?* to see their preferences and areas of curiosity. We hoped these answers would help tailor students' modelling experiences, leading to higher engagement in the future. The majority of the students mentioned the class diagram to be the most interesting. Interestingly, they provided various reasons for this:

> *"I found the class diagram the most interesting because it gave the best idea of how the program should be programmed. It gave a good visual representation of how the program should function and work"* — Student_C1
>
> *"Most definitely the class diagram. It think it will be very beneficial to know how it works. And to iterate through the process of making a system to update the requirements"* — Students_C2
>
> *"Class diagram, because in the class diagram you have to implement all the functions for each class"* — Student_C2

After class diagram, students find sequence diagram interesting.

> *"The sequence diagrams were really interesting; it was interesting to visualize how the user interacts with the system and what could go wrong"* — Student_C2

We noted a trend towards a more elaborate responses from the students. Notably, students provided more comprehensive insights into individual diagrams. Furthermore, there was an enhanced recognition of modelling as a thinking tool and how it contributes to long-term proficiency and its application in software development.

> *"Different ways to think about the problem and modelling them likewise"* — Student_C1
>
> *"I think the most beneficial aspect of what I learned from UML modelling is to think more like how a software engineer. When developing a software you have to think of every aspect and constantly plan for everything by making diagrams, unit testing and also get to now the target audience better"* — Student_C2

4.2.2 Challenges of Modelling. We asked students, *what challenges or difficulties have you encountered while studying UML?*

Many students were overwhelmed with the UML notation and its vast complexity.

> *"There is no one right way of executing the development process and we have to find what method is best for us"* — Student_C2
>
> *"Understanding the different rules, like the dotted lines and all of those small parts"* — Student_C1
>
> *"There seems to be different types of "syntax". Maybe I was looking at the wrong things, but when looking up images of diagrams they often looked different to each other"* — Student_C2

We noticed several syntactical and semantic mistakes in students' assignments. While syntactical errors were somewhat reduced in the group assignments, students still struggled with semantics. As students were free to choose any tool they preferred, we received varying layouts for the same problem and its solution model. Nonetheless, we observed that students had difficulty with open-ended problems where they needed to select elements for the model, such as different classes for a class diagram or various types of validation required to perform a basic registration service through a sequence diagram. We realised that students needed support in understanding the appropriate level of detail to include in a model. Students raised specific concerns about how much information a model should contain.

> *"Sometimes it was difficult to realise how detailed the models were supposed to be, that was the hardest part"* — Student_C2

5 Discussion

Our research findings highlight the positive impact of the four recommendations on software modelling practices within the university environment. However, despite the assistance provided by these recommendations, challenges persist regarding students' overall grasp of software modelling, particularly in terms of applying this knowledge beyond the confines of the course. It is crucial to explore what students are modelling and what knowledge they acquire from these courses in light of the implemented recommendations.

5.1 What do students model?

Giving students multiple assignments with different diagram types is useful, but stating a clear purpose is necessary for them to understand modelling. Student assignments show that students need help understanding the correctness of a diagram. Students emphasise syntax, which is concerning as most of the modelling experts argue the importance of semantics over syntax. Two factors triggered this aspect among students. First, students perceive correctness primarily as syntactical correctness. Second, the concept of semantics remained unclear to them.

> *"How to use the notations correctly based on our vision of the program because I feel like it is so easy to make a mistake. "* — Student_C1

However, students need help understanding a diagram's semantics regarding a problem description.

> *"The difficulties I have encountered is learning how to understand the analysis of the diagram and try to figure out if you have gather sufficient information. "* — Student_C1

Students often struggle with *unclear expectations* in modelling courses [5]. This arises firstly from students encountering modelling diagrams for the first time and secondly from the overwhelming nature of UML. Learning and practising diagrams within a limited time frame can only provide limited assistance to students in mastering modelling. Since the syntax of UML is complex, their learning tends to be limited to these concerns, not moving beyond mastery of syntax.

In analysing assignments from both C1 and C2, we observed common syntactical mistakes. For instance, incorrect associations between classes were made by 95 percent of students in C1 and

94 percent in C2 during their initial attempts at drawing class diagrams[7]. Similarly, mismatches between direct and return messages in sequence diagrams were seen with 73 percent of students in C1 and 68 percent in C2 making this mistake in their first attempts[8]. While syntax improvements were seen in the second attempts at drawing these diagrams, snytactical issues persisted even in subsequent attempts.

In both C1 and C2, during the group assignments on modelling, students were required to review each other's initial diagrams, received feedback, and then collaboratively drew a second diagram. This approach helped their comprehension of various diagrams and their basic notations. However, as the diagrams all addressed to the same problem domain, students were unable to understand different semantics from other examples. In a focus group study, a student mentioned:

> *"I don't think we ever saw like an actual, fully-formed diagram or something, like, for another problem domain"* — Student in the focus group

Even though students got the syntax and semantics right, sometimes, understanding what should be part of the diagram and what should be beyond the scope was difficult.

> *"I'm not sure how detail you guys want us to go, and, for example, we were doing a sequence diagram recently with a group assignment, and our group was really overthinking the sequence diagram, and we were going into a lot of details when we shouldn't have."* — Student in the focus group

With the grading rubric, we managed to break down the assignments into tasks, aligned purposes with each task and guided students on how to complete a modelling task. However, modelling is a creative process, and it is crucial for students to grasp what to model independently, albeit with the aid of guidelines. Consequently, we adjusted our recommendation R1 by incorporating diagram interpretation. We discuss this further in Section 5.3.

5.2 From university to industry: *How do we carry the knowledge?*

Students expressed that modelling helps them in various ways:

> *"Overall, I thought it was interesting to see these methods for modelling a system in different ways. I hadn't really given much thought to how systems should be mapped out before implementing them, but it's obviously crucial in any practical project"* — Student_C1

> *"I find that all of the diagrams that we have learned to make have helped me gain a better understanding of what it takes to make a software, and great to learn everything for the future "* — Student_C2

> *"I think the most beneficial aspect of what I learned from UML modelling is to think more like how a software engineer. When developing a software you have to think of every aspect and constantly plan for everything by making diagrams, unit-testing and also get to now the target audience better"* — Student_C2

However, several students expressed their doubts as well:

> *"if you look beyond the course, if you would apply this somewhere else now, either in a course or, you know, out in industry, do you feel like it would still be something that would help you to roughly know what you should do?"* — Student in the focus group

The primary challenge identified from our data is guiding students in the right direction and ensuring they retain their modelling knowledge for their future careers. Empirical data shows that lack of guidance or training poses a significant obstacle to the adoption of modelling in industry [12, 21, 31]. While the recommendations prove beneficial to students during their university studies, further guidance is needed to bridge the gap between academia and industry, especially in terms of training. Three aspects pose the greatest threat here: challenges regarding modelling tools, lack of realistic modelling problems, and training in courses similar to the industry.

Students are able to use modelling tools, but need substantial amount of training to understand the differences between tools and produce models efficiently [21]. In university curricula, students are often given the freedom to select their modelling tool, which is good as they can select a simple one. However, it has drawbacks as modelling tools often have different user interfaces and use different sub-sets of the UML (correctly or not), which need to be clarified.

> *"Because like we have, tools and the are four different types of dotted lines and you have like 16 different arrows, some filled, some are not, some are open other than not"* — Student in the focus group

Whittle and Hutchinson [34] find various mismatches between industry and educational modelling practice. The study reveals that, while modelling in education is very UML-centric, industry increasingly prioritises other notations, particularly at the meta-modelling level with bespoke domain-specific languages and code generators. In their study, Akundi et al. [3] conducted a two-phase study to check to which extent model-based systems engineering (MBSE) academic curricula in the US reflect industry workforce hiring requirements. The study provided a pathway to align university curricula with industry needs and strategies to mitigate adoption challenges. While UML can serve as a tool for entry-level courses to modelling, eventually education might have to bridge to industry practice. This applies to handling various tools and notations, as mentioned above, but also to move beyond syntactical issues and use models purposefully to engineer software systems. Note that this requires a certain maturity, and is therefore not suited for entry-level courses, where students have little knowledge of software engineering as a whole.

5.3 Refining Modelling Education Guidelines

We evaluated four recommendations and investigated two research questions: the effectiveness of these recommendations in improving modelling skills and how they influence students' perceptions of modelling. Our study found that the recommendations effectively enhanced students' modelling skills and led to a better learning experience, with improved communication between students and instructors. As students' perceptions of modelling improved, we gained deeper insights into their challenges in learning modelling. Based on the analysis of the responses to both research questions, we refined our recommendations as follows:

R1 Iterative Projects with Regular Feedback: It is essential for students to engage in more reading of diagrams, not just of diagrams created by other student groups, but also of diagrams produced by professionals. It is important to discuss strengths and completeness of a diagram given a problem description, and to teach student to explore what makes a good diagram.

> *"If I show you a class diagram for code generation, it will look ridiculous because it's so detailed, because it needs to be very precise and technical. Whereas, you know, I've seen company diagrams on their architecture, which were literally four boxes with a bunch of random arrows between them"* — Modelling expert in the focus group

Therefore, we propose that, in addition to iterative projects, we should include diagram interpretation as part of the modelling

[7] Data sourced from TA feedback for the assignments
[8] Data collected from TA feedback

assignment. This would serve as an essential precursor, allowing students to grasp important concepts before delving into diagram creation themselves. By starting with interpretation, students can develop a deeper understanding of the purpose and significance of each diagram component. Furthermore, it enhances communication between instructors and students regarding feedback, establishing clear expectations not only for assignments but also for students' participation in any modelling activities. Specifically, we recommend the following three course elements:

- Iterative Projects: Structure modelling courses around iterative projects as a foundational element.
- Model Interpretation: Emphasise the importance of model interpretation as the initial step before commencing modelling activities.
- Consistent Feedback: Provide ongoing and consistent feedback for each stage of the modelling process to facilitate student learning and growth.

R2 Limitation of Diagram Types: We observe from our findings that limiting the diagram types helps students, especially first-year students. The essential thing here is to show students why those particular diagram types are selected. Previous research shows the connectivity between class diagram and object-oriented programming [29]. Similarly, we observe that students can see the connection between class diagrams and object-oriented programming concepts, helping them to bridge new concepts to exisiting knowledge in programming. Further, we encourage instructors to anchor diagram learning with more theoretical or technical concepts. For example, a student mentioned,

> *"I thought the sequence diagram was an interesting way to look at users interacting with the system."* — Student_C1

Therefore, we propose to keep the recommendation as it is:

- Limited Modelling Diagrams: Limit the number of modelling diagrams covered in the syllabus to avoid overwhelming students. Provide a rationale for the selection of each modelling diagram type, explaining why they are relevant and beneficial for the course objectives.

R3 Grading Rubrics: Essentially, modelling is a tool for understanding a domain and planning or designing a system. In industry, these diagrams help comprehend problems and plan solutions. In a course, however, students learn syntactical aspects first, before learning their use. The grading rubric could focus initially on technicality and completeness, then progress to evaluating the effectiveness of the diagrams in addressing a specified purpose. This shift from syntax to purpose is crucial for preparing students for real-world applications. Also, rubrics can be a great tool of communication between TAs, TAs and students and between students themselves. We suggest to use rubric more into that direction. Therefore, in addition to a 'technical rubric', we propose to add am evaluation rubric for students to assess their modelling knowledge:

- Technical Rubric: Develop a technical grading rubric for assignments, focusing on evaluating the correctness and completeness of students' modelling diagrams.
- Evaluation Rubric: Introduce a second rubric for students to assess the effectiveness of the learned diagrams. Include checklists in this rubric to enable students to evaluate their learning against the course's learning outcomes.

R4 Use of Familiar Problem Domain: We advocate for instructors adopting a unified approach by selecting a single domain for the entire project, one that students are relatively familiar with. This approach not only fosters student engagement but also allows instructors to tailor the project to align with students' interests and the objectives of the course. Furthermore, it allows students to spend time on the concepts that are introduced in the respective course, namely software modelling, without too much cognitive load spent for understanding a new domain.

- Consistent Problem Domain: Ensure that all assignments in the course revolve around a single problem domain. Encourage instructors to involve students in selecting the problem domain by highlighting the importance of diverse domains and their unique information and development requirements.

6 Conclusion

We conducted a mixed-method empirical study to assess the impact of four modelling recommendations from related work, i.e., to use iterative modelling projects with regular feedback, to limit the amount of diagram types, to use detailed grading rubrics, and to use a familiar project domain throughout the course. We applied the four recommendations in two bachelor-level university courses and gathered data from students, teaching assistants, modelling and educational experts through surveys, interviews, detailed assignment analysis, and a focus group discussion. The results indicate the effectiveness of the recommendations in enhancing students' modelling skills and knowledge. However, we also identified areas for improvement, particularly in guiding students to apply their knowledge in real-world industry settings. As a result, we propose modifications to the recommendations, incorporating specific insights from our study to enhance students' modelling experience in university settings and for long-term effectiveness.

References

[1] Luciane TW Agner, Timothy C Lethbridge, and Inali W Soares. 2019. Student experience with software modeling tools. *Software & Systems Modeling* 18 (2019), 3025–3047.

[2] Luciane Telinski Wiedermann Agner, Inali Wisniewski Soares, Paulo Cézar Stadzisz, and Jean Marcelo Simão. 2013. A Brazilian survey on UML and model-driven practices for embedded software development. *Journal of systems and software* 86, 4 (2013), 997–1005.

[3] Aditya Akundi and Wilma Ankobiah. 2024. Mapping industry workforce needs to academic curricula–A workforce development effort in model-based systems engineering. *Systems Engineering* (2024).

[4] Omar Badreldin, Timothy Lethbridge, Arnon Sturm, Waylon Dixon, Abdelwahab Hamou-Lhadj, and Ryan Simmons. 2015. The effects of education on students' perception of modeling in software engineering. CEUR Workshop Proceedings.

[5] Shalini Chakraborty and Grischa Liebel. 2023. We do not understand what it says–studying student perceptions of software modelling. *Empirical Software Engineering* 28, 6 (2023), 149.

[6] Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastien Mosser, Richard F Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabi Taentzer, et al. 2018. Towards a body of knowledge for model-based software engineering. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 82–89.

[7] Gregor Engels, Jan Hendrik Hausmann, Marc Lohmann, and Stefan Sauer. 2006. Teaching UML is teaching software engineering is teaching abstraction. In *Satellite Events at the MoDELS 2005 Conference: MoDELS 2005 International Workshops Doctoral Symposium, Educators Symposium Montego Bay, Jamaica, October 2-7, 2005 Revised Selected Papers 8*. Springer, 306–319.

[8] Jane Frances Gilgun. 1992. *Definitions, methodologies, and methods in qualitative family research*. SAGE Publications, Inc.

[9] Fabian Gilson. 2018. Teaching software language engineering and usability through students peer reviews. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 98–105.

[10] Hassan Gomaa and L Burgueño. 2017. Teaching Software Modeling and Design.. In *MoDELS (Satellite Events)*. 543–546.

[11] Laure Gonnord and Sébastien Mosser. 2018. Practicing domain-specific languages: from code to models. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 106–113.

[12] Tony Gorschek, Ewan Tempero, and Lefteris Angelis. 2014. On the use of software design models in software development practice: An empirical investigation. *Journal of Systems and Software* 95 (2014), 176–193.

[13] Imed Hammouda, Håkan Burden, Rogardt Heldal, and Michel RV Chaudron. 2014. Case tools versus pencil and paper. In *ACM/IEEE 17th Int. Conf. on model driven engineering languages and systems–educators symposium*.

[14] James Hiller. 2016. Epistemological foundations of objectivist and interpretivist research. (2016).

[15] Sascha Kirstan and Jens Zimmermann. 2010. Evaluating costs and benefits of model-based development of embedded software systems in the car industry–results of a qualitative case study. In *Proceedings Workshop C2M: EEMDD „From code centric to model centric: Evaluating the effectiveness of MDD" ECMFA*.

[16] Jyrki Kontio, Johanna Bragge, and Laura Lehtola. 2008. The focus group method as an empirical tool in software engineering. In *Guide to advanced empirical software engineering*. Springer, 93–116.

[17] Maria Kutar, Carol Britton, and Trevor Barker. 2002. A comparison of empirical study and cognitive dimensions analysis in the evaluation of UML diagrams. In *Procs of the 14th Workshop of the Psychology of Programming Interest Group (PPIG 14)*. 1–14.

[18] Ludwik Kuzniarz and Miroslaw Staron. 2005. Best practices for teaching UML based software development. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 320–332.

[19] Ludwik Kuzniarz, Miroslaw Staron, and Claes Wohlin. 2004. An empirical study on using stereotypes to improve understanding of UML models. In *Proceedings. 12th IEEE International Workshop on Program Comprehension, 2004.* IEEE, 14–23.

[20] Grischa Liebel, Rogardt Heldal, and Jan-Philipp Steghöfer. 2016. Impact of the use of industrial modelling tools on modelling education. In *2016 IEEE 29th International conference on software engineering education and training (CSEET)*. IEEE, 18–27.

[21] Grischa Liebel, Nadja Marko, Matthias Tichy, Andrea Leitner, and Jörgen Hansson. 2018. Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice. *Software & Systems Modeling* 17 (2018), 91–113.

[22] Sharan B Merriam et al. 2002. Introduction to qualitative research. *Qualitative research in practice: Examples for discussion and analysis* 1, 1 (2002), 1–17.

[23] Kai Petersen and Cigdem Gencel. 2013. Worldviews, research methods, and their relationship to validity in empirical software engineering research. In *2013 joint conference of the 23rd international workshop on software measurement and the 8th international conference on software process and product measurement*. IEEE, 81–89.

[24] Parsa Pourali and Joanne M Atlee. 2018. An empirical investigation to understand the difficulties and challenges of software modellers when using modelling tools. In *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. 224–234.

[25] Eric J Rapos. 2018. We'll make modelers out of'em yet: introducing modeling into a curriculum. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 130–134.

[26] Rebecca Reuter, Theresa Stark, Yvonne Sedelmaier, Dieter Landes, Jürgen Mottok, and Christian Wolff. 2020. Insights in students' problems during UML modeling. In *2020 IEEE Global engineering education conference (EDUCON)*. IEEE, 592–600.

[27] Johnny Saldaña. 2015. *The coding manual for qualitative researchers*. SAGE Publications.

[28] Carolyn B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering* 25, 4 (1999), 557–572.

[29] Martina Seidl, Marion Scholz, Christian Huemer, and Gerti Kappel. 2014. *UML@ classroom: An introduction to object-oriented modeling*. Springer.

[30] Dave R Stikkolorum, Truong Ho-Quang, and Michel RV Chaudron. 2015. Revealing students' uml class diagram modelling strategies with webuml and logviz. In *2015 41st Euromicro conference on software engineering and advanced applications*. IEEE, 275–279.

[31] Marco Torchiano, Federico Tomassetti, Filippo Ricca, Alessandro Tiso, and Gianna Reggio. 2013. Relevance, benefits, and problems of software modelling and model driven techniques—A survey in the Italian industry. *Journal of Systems and Software* 86, 8 (2013), 2110–2126.

[32] James Vallino. 2006. If you're not modeling, you're just programming: modeling throughout an undergraduate software engineering program. In *International Conference on Model Driven Engineering Languages and Systems*. Springer, 291–300.

[33] Bernd Westphal. 2019. Teaching Software Modelling in an Undergraduate Introduction to Software Engineering. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. 690–699. https://doi.org/10.1109/MODELS-C.2019.00105

[34] Jon Whittle and John Hutchinson. 2012. Mismatches between industry practice and teaching of model-driven software development. In *Models in Software Engineering: Workshops and Symposia at MODELS 2011, Wellington, New Zealand, October 16-21, 2011, Reports and Revised Selected Papers 14*. Springer, 40–47.