How do we teach Modelling and Model-Driven Engineering? A survey

Federico Ciccozzi
Mälardalen University
Västerås, Sweden
federico.ciccozzi@mdh.se

Michalis Famelis
Université de Montréal
Montreal, Canada
famelis@iro.umontreal.ca

Gerti Kappel
CDP, TU Wien
Vienna, Austria
gerti@big.tuwien.ac.at

Leen Lambers
Hasso-Plattner-Institut
Potsdam, Germany
Leen.Lambers@hpi.de

Sebastien Mosser
Université Côte d'Azur, CNRS, I3S
Sophia Antipolis, France
mosser@i3s.unice.fr

Richard F. Paige
University of York
York, UK
richard.paige@york.ac.uk

Alfonso Pierantonio
University of L'Aquila
L'Aquila, Italy
alfonso.pierantonio@univaq.it

Arend Rensink
University of Twente
Enschede, The Netherlands
arend.rensink@utwente.nl

Rick Salay
University of Toronto
Toronto, Canada
rsalay@cs.toronto.edu

Gabi Taentzer
Philipps-Universität Marburg
Marburg, Germany
taentzer@informatik.uni-marburg.de

Antonio Vallecillo
Universidad de Málaga
Málaga, Spain
av@lcc.uma.es

Manuel Wimmer
CDL-MINT, TU Wien
Vienna, Austria
wimmer@big.tuwien.ac.at

ABSTRACT

Understanding the experiences of instructors teaching modelling and model-driven engineering is of great relevance to determining how MDE courses should be managed in terms of content, assessment, and teaching methods. In this paper, we report the results of a survey of 47 instructors in this field. Questions address course content, tools and technologies used, as well as positive and negative factors affecting learning outcomes. We analyse the results and summarise key findings with the potential of improving the state of teaching and learning practices. The survey is a preliminary effort in giving a structured overview on the state-of-the-practice within teaching modeling and model-driven engineering (from the point of view of the instructor).

CCS CONCEPTS

• **Social and professional topics** → **Software engineering education**; **Model curricula**;

KEYWORDS

Education, modelling, Model-Driven Engineering

MODELS '18 Companion, October 14–19, 2018, Copenhagen, Denmark

ACM ISBN 978-1-4503-5965-8/18/10…$15.00
https://doi.org/10.1145/3270112.3270129

ACM Reference Format:
Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sebastien Mosser, Richard F. Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabi Taentzer, Antonio Vallecillo, and Manuel Wimmer. 2018. How do we teach Modelling and Model-Driven Engineering? A survey. In *ACM/IEEE 21th International Conference on Model Driven Engineering Languages and Systems (MODELS '18 Companion), October 14–19, 2018, Copenhagen, Denmark.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3270112.3270129

1 INTRODUCTION

The sheer complexity of today's software has made modelling and Model-Driven Engineering (MDE) a pervasive discipline in software development [7]. While there have been several industry-wide studies of the *application* of MDE in practice (see for instance [5, 8]), little has been done to understand what is the state-of-the-practice in *teaching* modelling and MDE. Arguably, understanding the experiences of instructors teaching modelling and MDE is of great relevance to determine how MDE courses should be managed in terms of content, assessment, and teaching methods. While conveying to students the proper skills and expertise is crucial to let them pursue careers in software development, this area is also a key interface between research and teaching [1].

In this paper, we report the results of a survey of 47 instructors. Questions address course content, tools and technologies used, and positive and negative factors affecting learning outcomes. We analyse the results and summarise key findings with the potential for improving the state of teaching and learning practice. The work is meant to contribute to the definition of the state-of-the-practice of teaching modelling and MDE (from the point of view of the instructor). In particular, it may help in understanding the constructive alignment [3, 4] between course contents and learning objectives.

It is worth noting that this is an initial step in giving a structured overview on the state-of-the-practice. In particular, it has been developed in combination with the MBEBOK presented in [6], which proposes a set of fundamental concepts, terms and mechanisms that should constitute a Model-Based Software Engineering Body of Knowledge.

Structure of the paper. The paper is structured as follows. In Sect. 2 the survey is introduced. The next section presents the results, including general information, content, tools and technologies, and positive and negative aspects. In Sect. 4 a discussion about the results is provided. Finally, in Sect. 5 conclusions are drawn.

2 SURVEY

The main goals of this survey are to analyse the current state of modelling teaching, to better understand the distinctive aspects that characterise a successful course and what may not work, and to identify the typical learning outcomes and discuss them with respect to the needed ones. The survey questions are structured as follows:

General information. This is used to describe the context for each course and includes: name, level of students, whether it is elective or mandatory, the number of times the course has been given, the course format (e.g., lecture, seminar, etc.), number of credits, number of students, modes of assessment (exam, assignment, etc.), and learning outcomes.

Which aspects are predominantly covered? This identifies the focus of the course, whether it is predominantly about models for software engineering, language engineering or some other domain.

How models are used. This identifies how models are used as part of the teaching process – e.g., used for presenting a modelling method, developed by students in assignments or tests, etc.

What students do with models. This identifies what specific activities students carry out with models – e.g., creating a statechart from natural language, verifying system properties, etc.

Tools and technology. This identifies the platforms, technologies, and tools used during the course.

Positive and negative aspects. This open-ended question elicits stories about what worked and what didn't in the course.

3 ANALYSIS OF RESULTS

In this section, the results of each survey question is illustrated and discussed.

3.1 General information

In most of the cases the course name refers to Model-Driven Engineering (or a variants of it) and Software Engineering, however different names are sometimes used as follows (frequency in parentheses):

- Model-Driven Engineering (20), including the variants
  - Model-Driven Software Engineering (2)
  - Model-Driven Software Development (5)
  - Fundamentals of Model-Driven Engineering (1)
  - Advanced Design of Software Architectures: Model-Driven Engineering (1)
  - Modelling and Meta-modelling (1)
- Software Engineering (8)

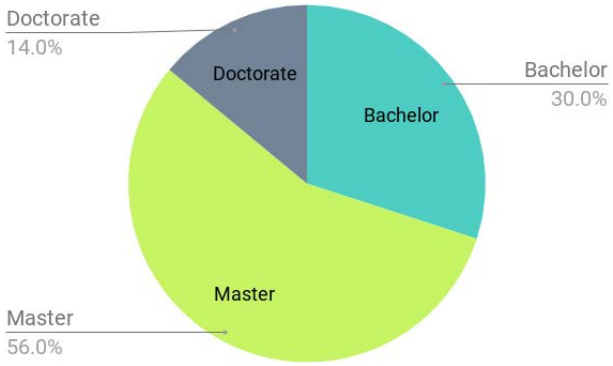


Figure 1: Level of students

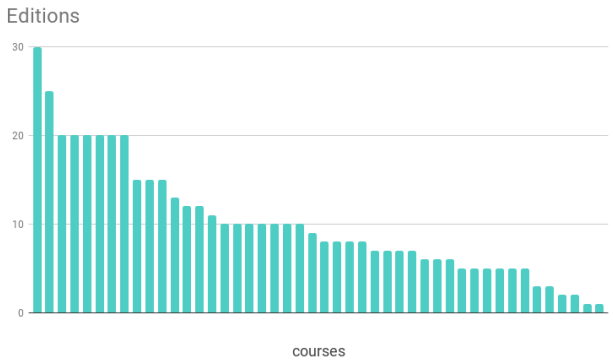


Figure 2: Editions

- Formal Methods (5)
- Software Modelling / Design (8)
- Software Architecture (3)
- Object-Oriented Programming / Programming (3)

Courses are delivered at the doctoral, master, and bachelor level as depicted in Fig. 1. It is worth noting that the majority of them are provided at the master level.[1] About half (52.8%) of the courses are elective, the rest of them mandatory (47.2%). The average number of course editions is 9,5 and according to the distribution in Fig. 2 it is possible to conclude that modelling and Model-Driven Engineering are well-established topics in Computer Science. In some cases, the number of editions is very high (up to 30): this is explained by the same course being given several times per year, i.e., in different semesters and/or at different levels.

Most of the courses are based on lectures, often in combination with laboratory activities. In one third of the cases, the course is delivered without laboratory. Besides lectures and laboratory, formats include seminars, colloquium, and tutorials, most of the times in some combination. The overall picture is given in Fig. 3.

The number of students for each course varies considerably. Almost half (47%) of the courses have up to 30 students, while the average number of students is 52 as plotted in Fig. 4. This is explained by the fact that courses given at the bachelor level can have a large number of students (in one case 300 students). The

[1] Some courses are given at more than one level, e.g., master and doctoral level.

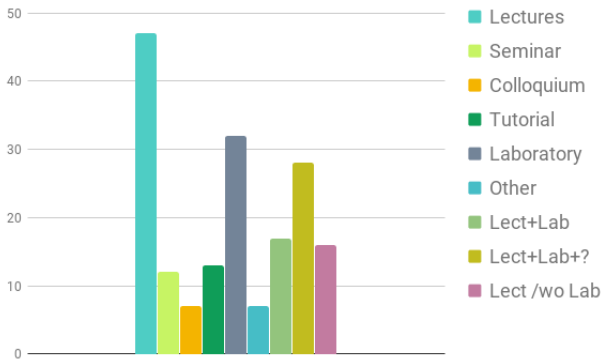


Figure 3: Course formats and their combinations

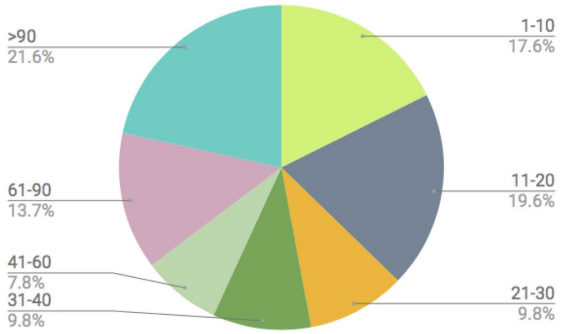


Figure 4: Number of students

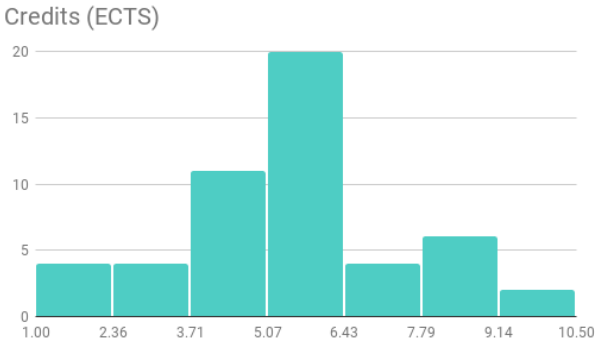


Figure 5: ECTS credits

number of credits is given in ECTS[2] (European Credit Transfer and Accumulation System) credits. A full study year normally consists of 60 credits according to this system. The distribution of credits among courses is given in Fig. 5, where the x-axis and the y-axis reports the number of credits and corresponding number of courses, respectively; a 6-credit course is the most frequent (interval 5.07-6.43).

Finally, most of the courses perform a student assessment that strongly focuses on tooling and technologies. Indeed, the survey shows that:

- 72% of the courses evaluate students by means of project assignments:
  - 84% of which are in combination with other modalities
    - home assignments
    - oral and written exams
  - 16% use projects only
- 78% by means of projects or home assignments
- 7% by means of oral and/or written exams

These results are presented in Fig. 6, where on the left-hand side the number of occurrences of all modes is reported, whereas on the right-hand side only modes comprehending project assignments are analysed.

The last question in the General Information section of the survey is about the *learning outcomes*. However, in order to better position this relevant aspect, the discussion is deferred to Sect. 3.3 after the analysis of responses about the topics covered by the course, as these two pieces of information are highly correlated as they might influence the constructive alignment (in the sense of Biggs [3]).

3.2 Course contents

The questions about the course contents are among the most significant ones in the survey. While in the general information section, the questions are about infrastructural (and unbiased) aspects, such as the average number of students, and the number of editions, the content section consists of questions that might convey some bias. In order to mitigate the problem, before discussing the individual findings, the questions are presented before the discussion of the corresponding responses.

Q1. Content / Which aspects are predominantly covered?

Describe the kind of models covered by the course.

A1. Models for Software Engineering (presentation of existing modelling languages and approaches for SE)
A2. Models for Language Engineering (incl. modelling language design, model transformation & model management)
A3. Models for other domains

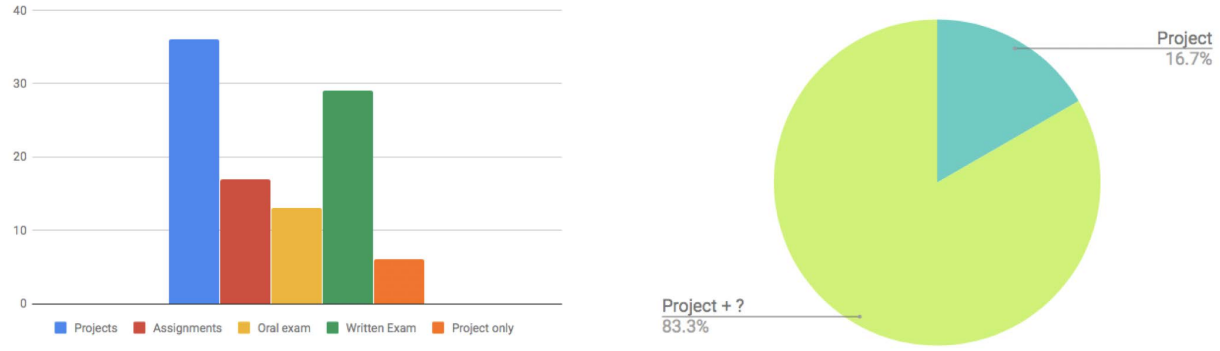
Figure 7 summarises the responses to question Q1. In particular, most of the courses adopt models for Software Engineering (49.1%) and Language Engineering (41.5%) with a residual part (9.4%) covering other application domains including robotics, IoT, e-learning, smart home, automotive and health. It is worth noting that the answers Q1.A1-2 might convey an epistemic bias as specific fields are suggested according to the authors' expertise. However, it is not difficult to see that the responses of other related questions, e.g., tools and technologies (see Sect. 3.4), are consistent with the results here.

Q2. Content / How models are used

Describe how models are used in the teaching process.

A1. As part of presenting a modelling method

[2] http://ec.europa.eu/education/resources/european-credit-transfer-accumulation-system_en



(a) All modes of assignment **(b) Project assignments**

Figure 6: Modes of assessment

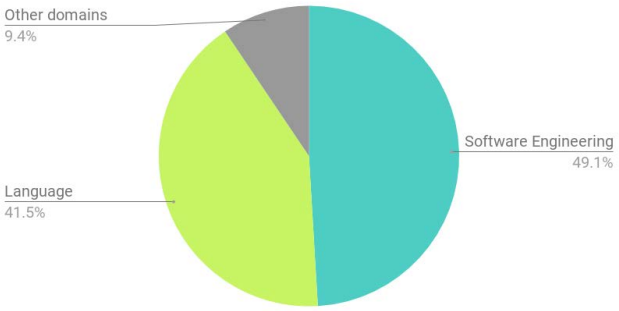


Figure 7: How aspects are covered

A2. Models are developed by students during their assignments or exam projects
A3. Other

In accordance with the results about the modes of assessment (see Sect. 3.1), in most of the courses (91%) "models are developed by students during their assignment or exam projects" while in more than the half of the courses (59%) models are used for "presenting the modelling method". In most of the cases both options have been selected.

Q3. Content / What students do with models

Please describe how the students use the models, e.g., i) Create statecharts from natural language req's to specify a system ii) Verifying system properties iii) Create temporal formulas for properties from natural language, etc.

What students do with models depends on the response to question Q1, i.e., whether the course refers to modelling in Software Engineering, Language Engineering, or another domain.

Software Engineering. Models are used for early design, communication, and documentation. They are typically created from requirements, including natural language scenarios. Both structural (e.g., class) diagrams and behavioural (e.g., state or sequence) diagrams are created and code is generated from models. OCL is often used for operation contracts or for expressing integrity constraints and invariants. Use case models (including semi-structured specifications) and executable statecharts (either simulated or generated) may be also employed.

Language Engineering. Students typically learn how to develop a modelling ecosystem consisting of

- metamodels, instances of such metamodels, model-to-model transformations, and model-to-text transformations (code generation); and,
- textual and diagrammatic editors.

The developed notations are mostly domain-specific, but class diagram- and activity diagram-like languages are also considered. When considered, requirements are in natural language. Some correctness of design models w.r.t. requirement models may be also requested. Finally, in few cases models are also obtained by reverse engineering code.

Other domains. Models are expressed mostly using (extended) UML, ADL or sometimes a DSL. In case mathematical expressions are needed, such as constraints, annotations are added. Requirements are mapped into problems to solve; these include quality problems and their solutions, and commonality or variability analysis. Domains can generally be categorised as application domains (depending on the assignment), quality domains (such as techniques for fault-tolerance, security), math (optimisation, etc.), computer science (to be able to map the synthesised models to the realisation platform), etc.

3.3 Learning outcomes

The information collected about learning outcomes has been analysed according to the following dimensions:

- Knowledge and understanding,
- Skills and abilities,
- Judgement and approach

The answers were not always consistently structured and often were given in an informal style. However, a certain regularity emerged especially concerning Software and Language Engineering. Not enough accurate answers were given concerning the other

domains; however, some fragmentary information is reported at the end of this section.

Knowledge and understanding
Software Engineering:

- describe how models can be used for design and analysis of a software system;
- describe how models can be used for specifying the behaviour of a software system;
- summarise how models can be used for documenting software systems;
- illustrate how models can fit into a software process consisting of analysis, design and implementation.

Language Engineering:

- explain the principles and the underlying concepts: model, metamodel, constraints, transformation, semantics, abstract and concrete syntax;
- explain the architecture of contemporary modelling frameworks;
- explain how domain specific modelling languages can be realised within a contemporary modelling framework;
- explain the basic concepts and techniques underlying the automated generation of (diagrammatic and textual) editors and environments.

Skills and abilities
Software Engineering:

- create software analysis models;
- create software design models;
- explain the functionality of a system with the help of analysis and design models;
- write reports with the help of software models;
- translate software models into executable code.

Language Engineering:

- construct domain specific languages, e.g., specify metamodels including syntax and semantics;
- define syntactic constraints using a constraint language;
- realise metamodels within a modelling framework;
- construct model editors within a modelling framework;
- create model validators within a modelling framework;
- specify model transformations and realise them within a modelling framework;
- apply the domain specific modelling approach to a case.

Judgement and approach
Software Engineering:

- judge how well software models relate to their real-life counterparts;
- identify which software models are appropriate for modelling a system;
- critically assess the quality of software models.

Language Engineering:

- select appropriate modelling technologies for a modelling tooling problem at hand;
- assess the applicability and limitations of model-driven engineering and tools to develop of software;
- judge the practical application of modelling and model management in realistic scenarios;
- discuss and document the construction and validation of models and extensions of supporting software tools.

Moreover, further elements emerged. More in detail, metamodelling techniques and process, domain-specific modelling, and UML profiling are widely used in language engineering courses; whereas UML is used in software engineering courses (e.g. software architecture, software design, and object-oriented design). In addition, executable UML models are used when reactive systems are considered; some courses combine modelling with agile software development; also models are used for test case generation. Categorical methods are used for formally characterising the typical model-driven concepts.

3.4 Tools and technology

There are different flavours of MDE and educators have a difficult choice to select the right variant for their course [1]. Options include picking an industrial strength modelling tool, using a tool specifically designed for educational use, or simply ignoring tools altogether in favour of pencil and paper. The choice is often highly dependent on the context, e.g., tools currently used by the instructor in active projects, already available expertise in her research group, availability of a repository of model or language exemplars, and so on. As a result of these factors, making a decision about the tools to be adopted is not easy and risk-free. Thus, analysing how tools and technologies are employed in teaching and to let students develop their assignments is of central significance for anyone interested in assessing the teaching method. Arguably, a contributory factor limiting the use of modelling is the technological cognitive distance due to idiosyncratic aspects and limited maturity of existing tools, besides the lack of appropriate educational resources and pedagogical techniques that introduce models in the classroom.

In our survey, instructors have been asked to provide a description of the platforms and tools adopted in their course by means of an open question, as follows

> **Q4. Which tools are used?**
> *Describe the platforms, technologies, and tools used during the course.*

The outcome is illustrated in Fig. 8 where for each tool the frequency is reported. For the sake of readability, the diagram reports only those tools with frequency higher than two, whereas the remaining tools and techniques are (in parenthesis the frequency):

- USE, GEMOC Studio, PlantUML, Ecore, Alloy, Epsilon, Emfatic, EGL, ETL, Ruby/RubyTL, Henshin (2);
- MetaDepth, CDO, Neo4EMF, Palladio, StarUML, Umple, mCRL2, TopCased, SDMLib, IntelliJ, UML, JET, QVTr, Graph Transformation, DPF, ATOMPM, MoDisco, Cucumber, ANTLR, Sismic, EMF Validation Framework, OclinEcore, GEF, Graphiti, MDEForge, EMFCompare, Gliffy, Argo IML, Eclipse JDT, DSLTrans, Eclipse UML Tools, MetaDONE (1).

The tools in the figure are classified according to platform: light green denotes EMF-based tools, gray denotes tools that can be used with EMF artefacts, for instance because they provide some form

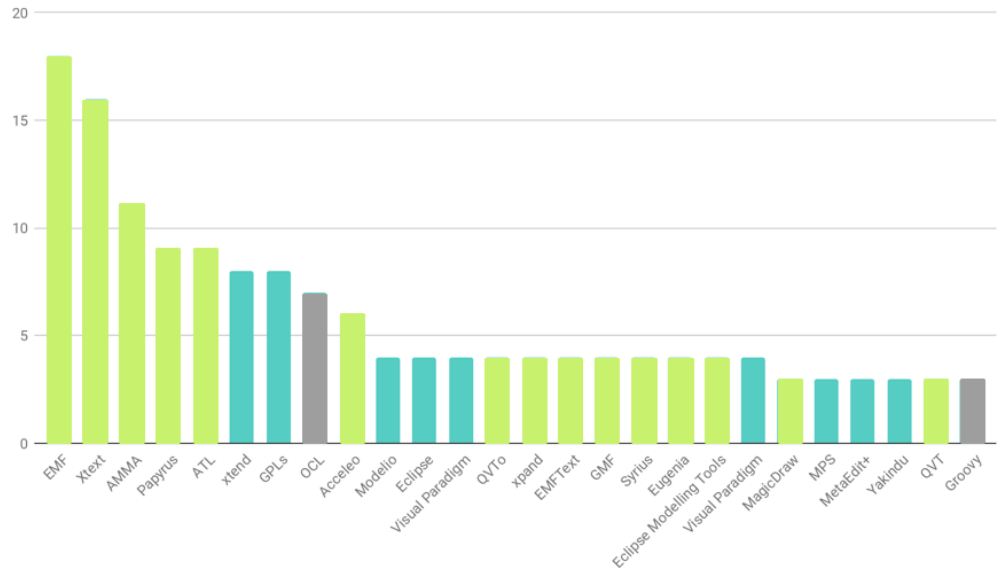


Figure 8: Modelling tools

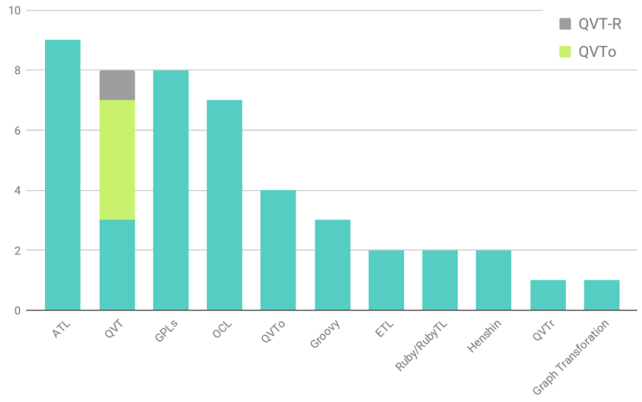


Figure 9: Model Transformation Languages

of interoperability; and finally in dark green are reported tools whose platform is distinguished from EMF. Interestingly, most of the tools are based on EMF, an open source platform that, despite the well-known difficulties in maintaining the needed components and plug-ins in a consistent state, is very popular among students. However, in contrast with other open source tools, e.g., MPS, EMF is community-based and part of the overall Eclipse project, that has surely provided traction to its adoption. As a rule, proprietary tools, like MetaEdit+, are less used.

3.4.1 Model Transformation Languages. If we restrict our attention to model transformation only, then the usage of corresponding languages is given in Fig. 9. Surprisingly, while well-known languages such as ATL[3], QVT[4], and ETL[5] are widely adopted, the second-most used means for teaching transformations are general-purpose languages. This can be (partly) explained by the need of transforming software models into analysis models in software engineering-based courses. In fact, most of the time the analysis notations, e.g., queuing networks, are not formalised by means of metamodels and therefore adopting a general-purpose language is among the options.

3.4.2 Concrete syntax. The tools adopted for defining concrete syntaxes and the related editors are almost equally divided in textual (58,8%) and diagrammatic (41,2%), although instructors have a preference for textual notations. As illustrated in the chart in Fig. 10, Xtext[6] is the most used tool, followed by another textual tool[7]. Then, the visual tools GMF[8], Sirius[9], and Eugenia[10] follow with the same percentage among them. While the situation for Xtext is pretty evident, as this represents one of the most well-documented projects (see e.g. [2]) for textual notations, the landscape of tools for diagrammatic syntaxes is less evident. We expect that in the next few years Sirius will emerge and that the reason why GMF is still relatively diffused is due to its legacy and to the difficulty in keeping technologies up to date.

3.5 Positive and Negative aspects

We included an open question at the end of the survey asking for both positive and negative aspects or experiences with respect to each course, as an open question.

Q5. Positive and negative aspects

Can you report on what worked and what didn't work in your course?

[3]https://www.eclipse.org/atl/
[4]https://www.omg.org/spec/QVT/
[5]https://www.eclipse.org/epsilon/doc/etl/
[6]https://www.eclipse.org/Xtext/
[7]https://github.com/DevBoost/EMFText, at the time of writing the official website http://www.emftext.org is unreachable.
[8]https://www.eclipse.org/gmf-tooling/
[9]https://www.eclipse.org/sirius/
[10]https://www.eclipse.org/epsilon/doc/eugenia/

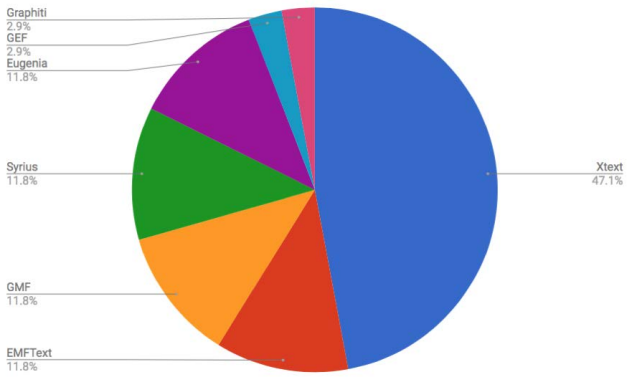


Figure 10: Concrete Syntax

From the gathered responses, we extracted positive and negative aspects. Then we looked for common themes occurring in each of the two categories. For the positive aspects, we came up with six common themes **P1-P6**, as depicted in Table 1. Among them, the most common positive aspect appears the effectiveness of using project-oriented and hands-on instead of classroom-based learning. One of the respondents states that *"[..] The project phase is a huge success once students understand that they are building tools for software developers (may it be general purpose or domain specific) [..]"*.

For the negative aspects we identified four common themes **N1-N4**, as described in Table 2. Among them, the most common issue is related to tools, being either too immature or complex to use. In this matter, one of the respondents states *"[..] Tools are an issue and it is not easy for students to detach models (and modelling) from what they are used to, code (and programming) [..]"*. Another one says *"[..] Tooling is not mature enough, students often got stuck, documentation is bad, [..]"*. This probably does not come as a big surprise, since there has been recently a general feeling in the community about the necessity to put a dedicated effort towards the conception of educational modelling tools (see e.g. Papyrus for Education[11]).

A question that arose when looking at the results is whether we could detect any relationship across positive and negative themes. We focused then on two types of relationship. The first one is "positive theme helps solve problem expressed by negative theme". This relationship is evident for P3,P6→N2 and P2,P4,P5→N3. In the first one for instance, focusing on very simple examples to explain theoretical aspects of modelling and gradually increasing the level of notions' complexity (P3) together with reflecting on existing modelling artefacts (P6) can help mitigating the difficulty of students in understanding the notion and importance of abstraction (N2). The second type of relationship is "negative theme hampers realisation of positive theme". This occurs with N1→P1,P2,P5 and N4→P1,P6. In the first case, immature or too complex tools (N1) hinder project-oriented and hands-on lectures/courses (P1) as well as first-hand experience with modelling benefits, such as code generation (P2) and model execution (P5). We cross-analysed the extracted data and found out that these relationships are basically confirmed by the survey responses. More specifically, there are only a few cases (at most 4 out 47 responses) in which P2,P4,P5→N3, N1→P1,P2,P5 and N4→P1,P6 are partially contradicted by the data extracted from the responses.

[11] https://wiki.eclipse.org/Papyrus_for_Education

Positive theme	Number of occurrences	Description
P1	13	Project-oriented or hands-on is more effective than classroom-based learning
P2	7	Code generation helps learning process and students see direct benefit
P3	5	Starting simple with both examples and theory and progressively getting more complex helps to overcome learning difficulties.
P4	3	Models related to familiar needs (e.g. communication, analysis) are easily seen as beneficial
P5	2	Executable models boost understanding of modelling benefits
P6	2	Studying existing modelling artefacts first helps understanding basic modelling concepts

Table 1: Common positive aspects

Negative theme	Number of occurrences	Description
N1	11	Tools are lacking, not mature enough or too complex to use
N2	10	Students have difficulties with understanding abstraction; modelling is conceived to be too different from programming
N3	9.5	Purpose of models is unclear
N4	7	Modelling languages or model transformation languages not precise or not powerful enough for their intended purpose

Table 2: Common negative aspects

4 DISCUSSION AND FUTURE WORK

Threats to validity. *Construct validity:* The participants of the survey were instructors, not students. It is therefore important to notice that we measured more and less successful aspects of a modelling course from the point of view of an instructor. *Internal validity:* In question Q1 the prescribed answers could have led to misleading results, since some of the modelling courses might not have a clear focus on either software engineering or language engineering. As all other questions allowed open answers, we do not see a similar bias there. In general, the questions in the list are largely independent of each other, so that the order of the questions should have no influence on the results. *External validity:* The list of questions in the survey was set up by the diverse group of authors

of this paper, being experienced in teaching modelling and model-driven engineering courses, as a joint effort. The questionnaire was tested and refined through a pilot survey. More specifically, we sent the questionnaire to a pilot set of 10 instructors prior to sending it to the actual respondents. This group of authors also came up with a list of 109 invitees to participate in the online survey. All invitees are knowledgeable in the community for their teaching efforts in modelling and model-driven engineering. In the end, 47 of these invitees participated in the survey. We cannot claim completeness w.r.t. all modelling and model-driven engineering courses being taught, but we believe to be able to have reached a representative part with our survey.

Discussion of current results. The landscape of modelling courses and their formats is very diverse. We could observe a strong focus on tooling and technologies for student assessment. Project assignments are typically appropriate to assess learning outcomes related to skills and abilities. However this raises the question whether the learning outcomes listed under "knowledge and understanding" or also "judgement and approach" are taken care of sufficiently by emphasising this type of assessment. The focus on tools and technologies might obfuscate the understanding of underlying principles of modelling and model-driven engineering. Since tools and technologies will continuously change, it seems important to align them with these principles. An interesting initiative that might help in this direction is the definition of a body of knowledge for model-based software engineering (MBEBOK) [6]. The list of learning outcomes compiled via this survey seems quite comprehensive already and is probably a good source of inspiration for instructors interested in designing a new course. The question remains how comprehensive this list is; therefore we propose to compare these results with the topics gathered in the MBEBOK as soon as it is publicly available. If particular learning outcomes observed through this survey do not occur in the MBEBOK, this might be a good starting point for discussion. The other way round, topics covered by the MBEBOK should be addressed by the learning outcomes observed in the current modelling course landscape. Finally, the learning outcomes in our survey relate either to software engineering or language engineering. The relevance of this dichotomy and assuming it in the context of teaching can be an interesting point for discussion also within such a broader initiative.

Correlating results of different questions. In Sect. 3 we analysed the results of each question individually; here we consider correlations between answers of different questions. We found that 2/3 of modelling courses with a focus on software engineering were Bachelor level while almost all language engineering courses were Master level. This makes sense given that the latter is a more advanced and specialised topic. We then asked whether negative/positive themes occur more often with software engineering or language engineering courses. Theme P3 (start simple and progress gradually) occurred almost exclusively for language engineering and this may be because the abstraction level is higher than for software engineering. Themes P1 (project-orientation) and N4 (languages not powerful enough) are twice as prevalent for software rather than language engineering while N1 (tools not mature enough) is twice as prevalent for language rather than software engineering. Thus, there is a support weakness in both areas but for software engineering it is with modelling languages while for language engineering it is with the tooling. We did not find significant correlations with other themes.

We investigated N1 further to understand how the top tools used fared. We found that about 30% of courses that used EMF, OCL or ATL and almost 50% of courses using Xtext, Acceleo or Xtend identified N1 as an issue. These results seem consistent with the fact that older tools would be more mature. A surprise was that 62% of courses using Papyrus cited N1.

Finally, we explored other potentially interesting correlations such as: If the purpose of models is unclear (N3), what did the students do with the models? Is the number of credits for a course correlated with the number of students? Do courses have similar characteristics when they are aiming for similar learning outcomes?, etc. but were unable to detect significant correlations.

5 CONCLUSIONS

We have analysed and discussed the results obtained by distributing a survey on the current state of modelling teaching to a number of instructors. While most of the responses do not expose any surprising aspect, the analysis shows the prevalence of assessment methods focusing on tooling and technologies. This can be explained by the great strides made in the development of modelling tools and techniques over the last few years. However, this has to be seen in a more general context keeping in mind the constructive alignment between the course contents and expected learning objectives. In this respect, the MBEBOK [6] might be a useful tool for orienting lecturers in the ways of designing a modelling course.

Acknowledgements. We would like to thank the reviewers for their valuable comments and suggestions. We are also grateful to all the instructors who responded to our invitation and contributed to the survey. This paper has been partially funded by the following research projects and grants: Spanish Research Project TIN2014-52034-R, by the Austrian Research Promotion Agency (FFG) via the Austrian Competence Center for Digital Production (CDP) under the contract number 854187, by the Austrian Federal Ministry of Science, Research and Economy and the National Foundation for Research, Technology and Development, and the Knowledge Foundation (KKS) through the MOMENTUM project.

REFERENCES

[1] Seiko Akayama, Birgit Demuth, Timothy C Lethbridge, Marion Scholz, Perdita Stevens, and Dave R Stikkolorum. 2013. Tool Use in Software Modelling Education.. In *Educators Symposium, MoDELS.*

[2] Lorenzo Bettini. 2016. *Implementing domain-specific languages with Xtext and Xtend.* Packt Publishing Ltd.

[3] John B Biggs. 2011. *Teaching for quality learning at university: What the student does.* McGraw-Hill Education (UK).

[4] Birgit Demuth. 2016. Constructive Alignment in Teaching Modeling. In *Educators Symposium, MoDELS.*

[5] Davide Di Ruscio, Richard F Paige, and Alfonso Pierantonio. 2014. Guest editorial to the special issue on success stories in model driven engineering. *Science of Computer Programming* 89, PB (2014), 69–70.

[6] Gerti Kappel Leen Lambers Sebastien Mosser Richard F. Paige Alfonso Pierantonio Arend Rensink Rick Salay Gabi Taenntzer Antonio Vallecillo Federico Ciccozzi, Michalis Famelis and Manuel Wimmer. 2018. Towards a Body of Knowledge for Model-Based Software Engineering. In *Proceedings of the MODELS Educators Symposium co-located with the ACM/IEEE 21st International Conference on Model Driven Engineering Languages and Systems (MODELS 2018), Copenhagen, Denmark, October 14-19, 2018. CEUR Workshop Proceedings.*

[7] D C Schmidt. 2006. Guest Editor's Introduction: Model-Driven Engineering. *Computer* 39, 2 (2006), 25–31.

[8] Jon Whittle, John Hutchinson, and Mark Rouncefield. 2014. The state of practice in model-driven engineering. *IEEE software* 31, 3 (2014), 79–85.