Software and Systems Modeling
https://doi.org/10.1007/s10270-024-01192-y

EXPERT VOICE

Requirements for modelling tools for teaching

Jörg Kienzle[1,2] · Steffen Zschaler[3] · William Barnett[3] · Timur Sağlam[4] · Antonio Bucchiarone[5] · Silvia Abrahão[6] · Eugene Syriani[7] · Dimitris Kolovos[8] · Timothy Lethbridge[9] · Sadaf Mustafiz[10] · Sofia Meacham[11]

Received: 27 March 2024 / Revised: 5 June 2024 / Accepted: 13 June 2024

Abstract
Modelling is an important activity in software development and it is essential that students learn the relevant skills. Modelling relies on dedicated tools and these can be complex to install, configure, and use—distracting students from learning key modelling concepts and creating accidental complexity for teachers. To address these challenges, we believe that modelling tools specifically aimed at use in teaching are required. Based on discussions at a working session organised at MODELS 2023 and the results from an internationally shared questionnaire, we report on requirements for such modelling tools for teaching. We also present examples of existing modelling tools for teaching and how they address some of the requirements identified.

Keywords Modelling · Education · Tools · Requirements

Communicated by Bernhard Rumpe.

✉ Steffen Zschaler
Steffen.Zschaler@kcl.ac.uk

Jörg Kienzle
Joerg.Kienzle@mcgill.ca

William Barnett
Will.Barnett@kcl.ac.uk

Timur Sağlam
timur.saglam@kit.edu

Antonio Bucchiarone
bucchiarone@fbk.eu

Silvia Abrahão
sabrahao@dsic.upv.es

Eugene Syriani
syriani@iro.umontreal.ca

Dimitris Kolovos
dimitris.kolovos@york.ac.uk

Timothy Lethbridge
timothy.lethbridge@uottawa.ca

Sadaf Mustafiz
sadaf.mustafiz@torontomu.ca

Sofia Meacham
smeacham@bournemouth.ac.uk

[1] ITIS Software, University of Málaga, Málaga, Spain

[2] School of Computer Science, McGill University, Montreal, Canada

[3] Department of Informatics, King's College, London, UK

[4] KASTEL, Karlsruhe Institute of Technology, Karlsruhe, Germany

[5] FBK, Fondazione Bruno Kessler, Trento, Italy

[6] IUMTI, Universitat Politècnica de València, Valencia, Spain

[7] DIRO, Université de Montréal, Montreal, Canada

[8] Department of Computer Science, University of York, York, UK

[9] School of Electrical Engineering and Computer Science, University of Ottawa, Ottawa, Canada

[10] Department of Computer Science, Toronto Metropolitan University, Toronto, Canada

[11] Department of Computing and Informatics, Bournemouth University, Poole, UK

1 Introduction

Modelling is a crucial aspect in software development and beyond [71]. It enables designers and engineers to efficiently explore the design space, and provides stakeholders with suitable representations of the system under study. Models are crucial in helping all involved parties understand, analyse, and design complex (software) systems. For the modelling benefits to materialise, the aforementioned activities have to be supported by modelling tools.

However, most of the academic and industrial tools currently available are not ideal for teaching modelling [12,

Published online: 09 July 2024

22]. Academic tools suffer in general from low maturity and robustness, and are difficult to install and maintain due to accumulated technical debt. This is mainly due to the fact that developing stable and usable (modelling) tools requires significant development effort and investment, and unfortunately this effort is usually accompanied by too little reward in terms of academic credit.

Industrial tools, on the other hand, come often with prohibitively high pricing and complexity, and no out-of-the-box support for teaching-related features, e.g. support for online collaboration, automated grading, and pedagogical feedback adapted to the student's level.

Motivated by this situation, we organised a 1-day working session on modelling tools for teaching (MTT) at the 26th International Conference on Model-Driven Engineering Languages and Systems.[1] From the discussions that day it is clear that our community believes that there is a real need for modern, intuitive MDE tools dedicated to teaching that are capable of demonstrating that models are highly beneficial development artefacts. We need tools that can inspire our students to use MDE for real, i.e. to drive other development activities, as opposed to simply creating pretty drawings to please the teacher. Because of the limited resources available for developing MTTs, we can only succeed if we work towards a common infrastructure for tools that we can collaboratively maintain and extend.

As a first step in that direction, we elaborated in our working session a set of requirements for MTTs. We present them in this paper organised as follows. Section 2 presents a brief summary of our working session and explains the questionnaire we sent out to the modelling community after the workshop. Section 3 lists the modelling-related capabilities that we envision are important for MTTs. Section 4 presents what we deem important pedagogical functionality that MTTs should provide. Section 5 discusses some technical requirements on MTTs. Section 6 concludes the paper by presenting some examples of existing MTTs and how they relate to some of the requirements discussed.

2 Methodology

We briefly describe the methodology used to elicit the requirements presented in this paper.

2.1 Workshop summary and pilot survey

The purpose of the 1-day working session at MODELS 2023 was to engage in productive discussions regarding the requirements and necessary infrastructure for MTTs. Targeted invitations were sent to research groups actively involved in developing tools used for teaching computer science in undergraduate or graduate classes.

In the first session, participants were allotted five minutes each to present the MTTs they were working on, focussing on their teaching-related aspects. Subsequent sessions, attended by an average of 25 people, were dedicated to brainstorming among attendees.

At the conclusion of the workshop, all participants were surveyed regarding their perceptions of the importance of tools, languages, features, and educational practices discussed earlier. The survey aimed to serve as a prototype for wider circulation within the community. Despite being produced within a half-hour timeframe, the survey yielded valuable insights:

- Participants expressed a keen interest in teaching using class diagrams and state machines, alongside other model types albeit with lesser importance.
- Important attributes for modelling technology in teaching included being free, being user-friendly, having multi-platform compatibility (including having a web-based version), having a comprehensive user manual, having a library of examples, being reliable, having performance analysis capabilities with feedback, having fast response time, being capable of code generation, and having a textual interface available.

Post-workshop, reflection on the prototype survey results led to the revision of questions and the preparation of a formal survey for broader circulation, as discussed in the following section.

2.2 Survey conducted following the workshop

A subgroup of the authors refined the pilot survey[2] over the course of 3 months following the workshop. Ethics approval was obtained to circulate the revised survey by the end of January 2024. Targeted sampling methods were employed for distributing the survey using the following steps:

1. Each workshop attendee was tasked with ensuring that the survey reached relevant individuals within their institution, including themselves.
2. Additionally, participants were encouraged to reach out to colleagues in other institutions within their geographic region and beyond.
3. The survey was also disseminated through the LinkedIn and X accounts of several authors.

[1] https://modellingtoolsforteaching.github.io/.

[2] The questions and answer options are openly available from the King's College London research data repository, KORDS, at https://doi.org/10.18742/25429270.

Table 1 Summary of survey questions about modelling tools for teaching (MTT)

Question	Theme
1	Confirmation of consent and teaching experience
2	Number of years they have been teaching
3	Extent to which they teach software modelling
4	Satisfaction with current modelling tools for teaching
5	Improvement areas for MTT (open ended)
6	Importance of teaching specific modelling languages at undergraduate level
7	General user experience attributes with MTT (nine items)
8	Editing and collaboration features of MTT (five items)
9	Language and language-manipulation features of MTT (ten items)
10	Analysis features of MTT (five items)
11	Transformation features of MTT (seven items)
12	Platform capabilities for MTT (nine items)
13	Business and economic issues related to MTT (five items)
14	Languages they would like the tool to generate
15	Teaching practices an MTT should support (13 items)
16	Modelling tools currently used (16 items)
17	Academic rank (for demographic analysis)
18	Continent (for demographic analysis)
19	Gender (for demographic analysis)

By the end of February 2024, we had gathered 59 valid responses. Only one response was excluded due to incomplete answers to the questions. Given the richness and depth of the data collected, we have decided to publish a comprehensive analysis in a separate paper. In that paper, we will first present the detailed questions of the survey as described in Table 1, followed by the main insights obtained from the responses. Here, we provide a brief characterisation of survey respondents and then focus on some central themes.

We asked about the participant's general background and demographics:

- **Teaching Experience and Duration (Q2):** Respondents to the survey broadly have substantial experience teaching modelling: 30% of respondents said they have been teaching modelling for 20 or more years. A further 23.3% have taught modelling for 10–19 years and 25% for 5–10 years. Only two respondents said that they "do not teach modelling (yet)".
- **Extent of Teaching Software Modelling (Q3):** A significant portion of respondents teach software modelling to varying degrees, with 39% heavily involved, 54% moderately involved, and 7% teaching minimally.
- **Satisfaction with Current Technology (Q4):** Mixed satisfaction levels were observed among respondents regarding the current technology available for teaching software modelling, with no respondents indicating being very satisfied, 54% expressing satisfaction to some extent, and 36% indicating some level of dissatisfaction.
- **Demographics (Q17–19):** Participants were roughly evenly divided among full professors, associate professors, and assistant professors. Responses were 74 per cent from Europe; 16 per cent from North America, and 10 per cent from elsewhere. Respondents were 75 per cent male and 25 per cent female.

Question 5 was an open-ended question about areas of modelling that could be improved and features that were required. The respondents identified several areas for improvement in software modelling tools for teaching, including various user experience issues, collaborative editing support, executability, support for modelling for particular architectures or frameworks, analysis capabilities, a strong preference for a web-based interface, and concern about outdated tools, standards, and languages. Many of these will be discussed in detail in the remainder of the paper.

The responses to Question 6 on the importance of teaching certain modelling languages are presented in Sect. 3.1.1.

Questions 7 through 13 ask about the significance of 50 distinct qualities and features within the realm of modelling tools and languages, in the context of teaching modelling. Each item was subject to assessment on a six-point scale, where respondents were tasked to rating their perceived importance.

The scale encompassed the following criteria:

- **Harmful**: Signifying qualities or features that would introduce unwarranted complexity into the process, assigned a weighted score of -1.

- **Not needed/not something I would judge a tool by**: Reflecting aspects perceived as extraneous or irrelevant to the evaluation of a tool, assigned a weighted score of 0.
- **Good to have at a basic level**: Denoting qualities or features that are beneficial but not imperative, allocated a weighted score of 1.
- **Important to have at a moderate level**: Identifying qualities or features of moderate significance, granted a weighted score of 2.
- **Essential to have reasonably good capability for this**: Highlighting qualities or features deemed fundamental with a reasonable level of proficiency, assigned a weighted score of 3.
- **Critical**: Must be as extensive and good as possible, indicating qualities or features of utmost importance, demanding the highest level of capability, and assigned a weighted score of 4.

This evaluation framework aimed to discern the varying degrees of importance attributed to each quality and feature, thereby facilitating a nuanced understanding of their relevance in the context of teaching modelling. We will discuss the responses to these questions in Sect. 7 after we have qualitatively reported on the workshop discussion that introduced these requirements.

Question 14 asked about code generation, and what programming languages a tool should support. We will discuss these responses in more detail in Sect. 3.1.5.

Question 15 asked about teaching practices that should be supported (multiple responses were possible). The most important practices to be supported include having students build small models (97 per cent need this); using the tool live in class (81 per cent); supporting project-based learning (80 per cent); supporting problem-based learning (75 per cent); having students analyse and criticise models (75 per cent); and having students improve an existing model (71 per cent).

In Question 16, we asked about the tools respondents use in teaching modelling. The top modelling tools or tool types used by the participants were PlantUML, EMF/Ecore/Emfatic, plain drawing tools (such as Draw.io), Visual Paradigm, XText, Umple, formal methods tools, MagicDraw, Figma, Papyrus, and Sirius.

3 Modelling-capability-related requirements

This section presents a list of modelling-capability-related requirements that were identified as important for MTTs, organised into two categories that target distinct groups of students.

Modelling tools are tools specific to a particular modelling formalism. Students use them to learn how to model in one or several languages, learning their syntax, structural and behavioural semantics. Such tools often come with a dedicated debugger and various development services.

Language workbenches [31] are generic modelling environments to create and use arbitrary modelling languages, like the Eclipse Modelling Framework (EMF) [72]. Students use them to learn to develop domain-specific modelling languages, with an emphasis on metamodelling and grammar design. Although generic services are provided, students need to develop their own model transformations and code generators.

3.1 Teaching modelling with existing languages

In this section, we present requirements on MTTs related to teaching modelling with existing modelling languages.

3.1.1 Modelling language support

In our survey, we asked respondents "To what extent do you think each of the following modelling languages should be taught to undergraduates as a required part of a computer science or software engineering program?" Here, we report on their responses.

We asked respondents to rank a set of modelling languages on a scale from "Harmful" to "Critical". A detailed overview of the responses is shown in Fig. 1.

The top 4 modelling languages (based on average scores) are UML sub-languages: class diagrams (with overwhelming support for criticality), state machines, sequence diagrams, and use cases—possibly reflecting on the typical syllabus covered in the teaching of modelling in software engineering courses at the undergraduate level. Interestingly, this is followed by entity-relationship models and activity diagrams on places 5 and 6.

On the other end of the scale, we find formal methods models other than OCL, Petri nets, goal models, and feature models. Notably, formal methods models attracted seven votes for "Harmful", possibly reflecting a continuing perception that formal models are difficult to engage with, especially at the undergraduate level.

3.1.2 Textual interface support

Although software modelling is widely thought of as focussing on diagrams, there are also many textual modelling languages including OCL [56], USE [64] and Umple [49]. In fact, experience has shown that modellers find it very useful to be able to define their models textually as well as graphically [6].

Graphical languages have many advantages, such as taking advantage of two dimensions and arbitrary shapes, but there are a variety of benefits of using textual modelling lan-

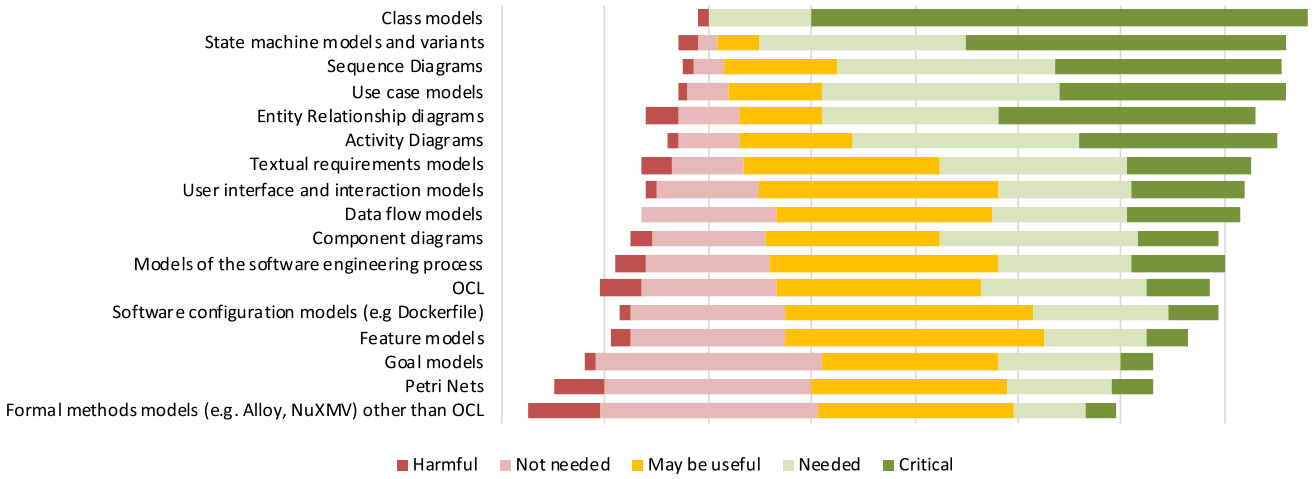


Fig. 1 Responses to Q6 on required modelling languages. This was a semantic-difference scale question and each coloured bar represents the percentage of responses for each possible answer, centred on "Not needed"

guages in teaching. Students are used to textual editors from coding; hence, editing text can be faster and more productive than editing graphics. Copying and pasting in particular can be easier. Furthermore, commenting and annotating can be easier in a textual format, since there are no layout constraints.

In the survey, we asked about the importance of having a textual interface available. Sixty-one per cent of respondents considered this important, essential, or even critical.

Proper support for textual modelling also requires support for syntax checking and validation, syntax highlighting, autocompletion, debugging, and navigation (the ability to quickly jump between different sections, e.g. by clicking references).

3.1.3 Support for consistency checking

Modelling tools for teaching should support conformance checking to ensure that students create model instances that adhere to the structure described in the language metamodel, making the modelling process for students less error-prone. Support for validation checks to make certain that all language constraints are upheld is also required (e.g. a state diagram has only one start state). Warnings and error messages should be generated to inform students of modelling mistakes.

3.1.4 Support for views and consistency between views

A fundamental aspect of MDE is the use of multiple models (views) to describe a given system [21]. These models may vary depending on the level of abstraction (e.g. from requirements models to architecture models) and also on the viewpoint (e.g. structural versus behavioural models) [10]. When teaching modelling, it is important that the student learns to define models that focus on different sets of concerns and that can be described using one or more languages (notations).

Maintaining the consistencies of the models manually is a tedious endeavour without dedicated support from the modelling tool [43]. In Abrahao et al. [1], model integration has been identified as a challenge affecting user experience with MDE tools. According to the authors, vertical and horizontal model integration (syntactic and semantic) is essential to ensure consistency. Therefore, a MTT should help students understand how the views of the systems are related. Ideally the tool should propagate changes in one view to the other views in order to maintain consistency, or alternatively the tool could signal a validation error.

3.1.5 Support for model execution/enactment/experimentation

Students need to be able to do something with the models they produce so that these models become more than "nice drawings". We differentiate three different purposes.

For comprehension Models can have different degrees of formality. While modelling with a low degree of formality is suitable for communication and documentation purposes, informal models lack the ability to be used as input to the software development lifecycle. It is therefore desirable to teach students to create models that are both understandable to stakeholders and that can be used to guide subsequent phases of software development. As several empirical studies show, models play a fundamental role in helping students and practitioners in understanding software specifications (e.g. software requirements [2], source code [68]). MTTs,

therefore, need to present models in ways that facilitate comprehension—including appropriate concrete syntax.

For production Another crucial use of models, specifically in industrial settings, is integrating them into software systems. Generating textual artefacts from models can be automated through template-based code generation [73] and this needs to be supported in MTTs. Especially source code generation enables students with a programming background to better engage in MDE and understand its concrete usefulness. Some models can be integrated into programs directly when they are executable, such as models developed in GEMOC studio [9].

In our survey, we asked respondents "If you would like code generation (model to text) to be part of your teaching process, which languages are the most important for a modelling tool to be able to generate?" 3 respondents replied "None", presumably indicating that they do not use model-to-text transformation in their teaching, but the overwhelming majority of respondents appears to include code generation in their teaching. Figure 2 provides an overview of these responses.

Java is clearly the leading target language for educators teaching model-to-text transformation (with 51 of 59 respondents choosing it), closely followed by Python (38 respondents). Surprisingly, SQL appears in third place, with C++ only some distance behind. Other languages mentioned included JavaScript, Kotlin, C, Go, Rust, and "textual artefacts that are not specifically related to a programming language".

For analysis The benefits of modelling can be further demonstrated during teaching by leveraging simulation, model checking, or traceability analysis methods. This will allow students to understand the behaviour of systems, verify the correctness of models, and validate the design. The need for simulation technologies is even more prevalent now with the Internet of Things and cyber-physical systems, since having the means to analyse and validate the system before incorporating changes into the running system is of critical importance [41]. Modelling environments for teaching should have integrated support for interactive simulation and (virtual) experimentation [53] enabling exploration and what-if analysis with parameter estimations.

Moreover, support for formal verification of models is essential for teaching modelling of critical systems [23, 77]. Model checking tools, such as SPIN [37] or UPPAL [47], could be integrated with MTTs to make them easily available and allow students to verify their models.

3.1.6 Support for an MDE process

When modelling is used throughout the software development lifecycle, i.e. from requirements engineering to architecture to design, implementation, and testing, teachers typically ask the students to follow a specific MDE process. An MTT that is process-aware could provide students with systematic guidance on which models to elaborate using which modelling language at what time, as well as help in maintaining horizontal and vertical traceability between the models. For collaborative tasks, support for identifying dependencies between the models would also improve coordination between teams and help in monitoring the progress of projects.

Moreover, it would be valuable for MTTs to provide means for generating skeletons of downstream models (e.g. deriving a partial design class model from a domain model or generating an activity diagram from a use case model) or even models at the same level of abstraction (deriving a partial domain model from a use case model). In addition to support forward engineering, tool support for reverse engineering to generate (partial) upstream models would be very useful for students to gain an understanding of how code is represented at higher levels of abstractions, and to learn how to refactor models and then generate other views, code, or documentation from them (see [11], for instance).

In the context of teaching MDE processes, tool support for explicitly modelling processes and traceability analysis would be valuable. Traceability information generated from software or business process model enactments can be used for dependency visualisation, origin tracking (for instance, backtracking from design to requirements artefacts), change impact analysis, change propagation as well as for streamlining and optimising processes (as seen in [34]).

3.2 Teaching the development of new modelling languages

In this section, we collect requirements on MTTs used for teaching students how to develop new modelling languages. Tools for the development of modelling languages are typically called Language Workbenches.

3.2.1 Support for different modelling paradigms

An important skill that students need to acquire during an advanced MDE course or a course on software language engineering is the development of a new modelling language, or adapting an existing modelling language and its modelling editor to fit a certain domain and/or certain needs.

For textual modelling, the tool should allow students to develop a grammar, specifying tokens and production rules

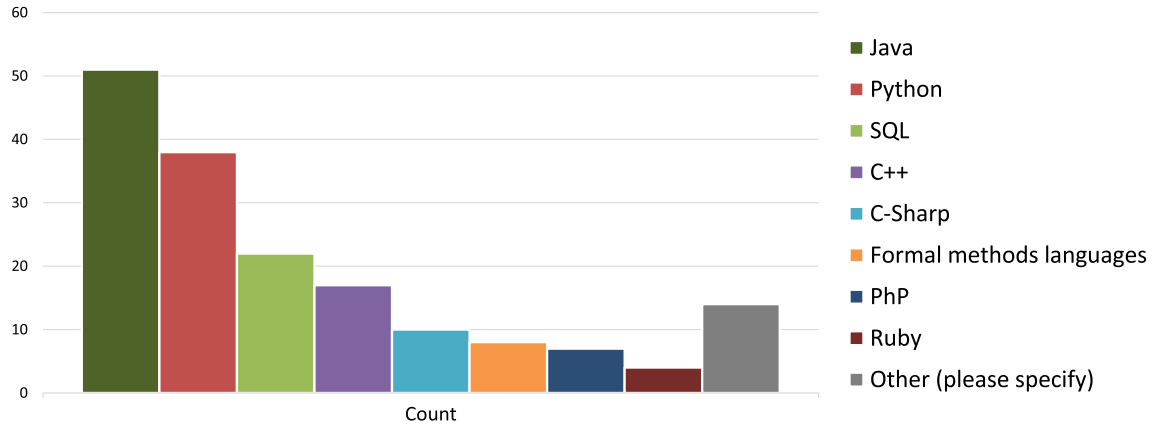


Fig. 2 Overview of responses to the question on desired target languages for model-to-text generation. The chart excludes the three responses indicating "None". Note that multiple responses were possible for this question

from which a parser can be generated as with Xtext.[3] To improve the user experience, students should be able to specify the styling properties of keywords or rules in the textual language, customising the font style, colours, and layout. From that, the MTT should generate an editor with common services as described in Sect. 3.1.2.

When a graphical notation is better suited for the modelling language, the MTT should allow individual metamodel elements to be mapped to their corresponding graphical representation. To this aim, different techniques could be supported, e.g. specifying the mapping through programming (GMF [61]), by annotating the metamodel (Eugenia [45]), or by explicitly modelling the concrete syntax (Sirius [78]).

3.2.2 Experimentation capabilities

When designing domain-specific languages, it is important for users to also be able to create sample models side-by-side to experiment with different design alternatives. To facilitate this exploratory style of language development, it is desirable for tools to provide coordinated visualisation capabilities for both models and metamodels (e.g. using object/class diagram notations or even by supporting custom graphical notations using metamodel annotations in the style of Eugenia [45]). It is also important that they tolerate (and highlight) inconsistencies in models as metamodels evolve.

3.2.3 Automated model management

As the crux of model-driven engineering is automated model processing, suitable tools must naturally provide support for composing, and executing a wide range of model management activities, e.g. model-to-text transformation, in-place and mapping model-to-model transformation, model validation, model comparison and merging, pattern matching, model migration, and model simulation.

Ideally, users should not be required to perform any configuration (e.g. to specify which program runs against which model, or which metamodel a model conforms to) by default. Offering configuration facilities to support arbitrary complex model management scenarios is not advisable as the tool then starts entering the realm of IDEs, which is undesirable. However, the tool should offer a smooth transition into an actual IDE, for example, by allowing the user to download their artefacts (model, metamodel, model management program) in a format that can be imported into an IDE (e.g. by auto-generating a `.project` file for Eclipse or a Gradle/Maven build file for compatible IDEs) or by providing a hyperlink that sets up and launches a web-based IDE (e.g. VS Code, Gitpod) with the user's artefacts.

4 Teaching-related requirements

In this section, we elaborate on specific pedagogical support we believe our MTTs should provide to support teaching. We first discuss the importance of aligning MTTs with educational terminology and standards as well as integrating them with traditional learning systems, and then split our discussion into modelling-specific support for the teachers and for the students.

4.1 Alignment with educational terminology and standards

Ensuring accessibility and quality of learning for a diverse audience necessitates the widespread utilisation of open educational resources (OER) [36, 57, 76] in all educational endeavours. In this context, the education system has used *competences* to organise learning and quantify personal and

[3] https://eclipse.dev/Xtext/.

professional growth. Proficiency *levels* categorising *competence items* enhance the *assessment process*, providing valuable insights into individual mastery.

To fortify this educational approach, there is a need to align with the competencies and skills elaborated in the most recent Computing Curricula of the ACM [18], but also with established models like O*NET,[4] ESCO,[5] and EntreComp,[6] ensuring that skill development resonates with industry standards. Competences, conceptualised as sets of interconnected *concepts*, encapsulate knowledge, principles, and practices crucial for achieving *learning goals*. The compilation of these elements in the *competence portfolio* offers a structured overview of skills, knowledge, and aspirations, guiding students through *competence requirements* and *acquired competences*, where teachers play a central role in ensuring a logical progression.

To this end, teachers define *learning paths*, segmenting them into focussed *learning fragments* centred on specific topics. These fragments, supported by associated *learning activities*, orchestrate a guided process facilitated by the *progress edges*. The progress edge introduces multiple outcomes from each learning activity, creating a dynamic learning experience that fosters personalised learning through diverse outcomes, tailored paths, adaptive feedback, learning flexibility, and heightened motivation.

4.2 Integration with learning environments

As suggested in [3], external tools, such as MTTs, should be integrated with traditional virtual learning environments (VLEs) to maximise the benefits for students and educators. This will enable the utilisation of common VLE provisions such as collaboration, communication, assessment, and feedback structures that have been proven necessary, effective, and pedagogically sound [17]. This will provide seamless integration within the students' environments and a more connected and insightful learning experience. Additionally, it not only makes learning more accessible, but also allows us to gather valuable information about how students interact with these tools. By collecting learning analytics [70] through these tools, educators can understand each student's progress, tailoring support to individual needs and facilitating further the individual learning paths. In essence, the integration of modelling tools into VLEs supports a personalised educational journey that incorporates familiar proven pedagogical methods.

[4] Occupational Information Network: https://www.onetonline.org/.

[5] European Skills, Competences, Qualifications, and Occupations: https://esco.ec.europa.eu/.

[6] European Entrepreneurship Competence Framework: https://ec.europa.eu/social/main.jsp?catId=1317.

4.3 Support for teachers

We first discuss requirements from a teacher perspective.

4.3.1 Modelling concepts and example library

Students learn well from concrete examples [54], but creating substantial high-quality examples requires a significant effort from teachers. Modelling tools for teaching should, therefore, have the ability to integrate with repositories of sample learning activities, which should be made available as open education resources (OERs). This could include: activities where students need to do the same activity with and without modelling [58]; an annotated repository of example models, transformations, and other modelling artefacts; or examples of industrial use of MDE.

4.3.2 Assessment support

Class sizes for software engineering courses continue to increase, and as a result, the assessment of students is becoming more and more of an issue. We imagine that an MTT could help a teacher discover or select modelling exercises from the aforementioned model repository for assessment. Better even, the MTT could generate modelling exercises, or variations of a given modelling exercise, to prevent cheating [33].

But most importantly, teachers need grading support. In our experience, the grading of models is often more time-consuming than the grading of code, because for a given situation there is in many cases more than one way of modelling it. MTTs should offer mechanisms for defining marking schemes, i.e. attaching of points to model elements, ideally in a language-independent way as done in [8, 38]. Automated grading algorithms could compare a student's solution with a teacher's solution, or, for executable models, run a set of model unit tests to determine the grade of the student. Ideally, the tool would also generate feedback for the students to let them know where and why they lost points. Again, it would be nice to have generic ways of providing feedback that are independent of a specific modelling language.

4.3.3 Plagiarism detection

Discouraging plagiarism in education is vital to uphold academic integrity, ensure fairness in evaluation, and teach students about ethical behaviour in both academic and professional environments. In large courses, however, manual inspection becomes impractical [27]. Plagiarism tends to be more prevalent in mandatory assignments, such as those in beginners' courses [59], which typically involve a higher number of students [16]. Thus, we identify the need for tool-

based solutions to tackle the problem at scale [67]. Plagiarism detection is well researched for code [55], however, not for modelling assignments [51]. They pose a unique challenge due to the fact that models typically operate at a higher level of abstraction, providing fewer details for detection. Furthermore, approaches designed for code rely on linearization, a process that is not trivial for models in general. Tool-based solutions should help teachers by identifying suspicious candidates while leaving final decision-making to the teachers to uphold ethical standards [48]. They also need to provide explainability and traceability in order for teachers to understand why a student submission is identified as suspicious.

4.3.4 Collaboration

We differentiate collaboration between teachers and between teachers and students.

Collaboration between teachers We identified at least two forms of teacher–teacher collaboration that would be helpful for developing complex learning activities in modelling.

In *sequential* collaboration, multiple teachers edit a shared learning activity, e.g. because one teacher reuses someone else's learning resource in the context of their own teaching, or because a team of teachers may be co-delivering the same course. This requires strong version-management support, made additionally challenging by the requirement to manage the consistency of a complete learning activity as individual parts are modified by different teachers over time. Ideally, an MTT would support authors of learning resources to review, comment on, and approve suggested changes by other teachers. Especially when resources are shared across institutional boundaries, a lightweight and asynchronous process of proposing changes to existing materials may facilitate better-maintained resources than an environment characterised by a "fork-and-forget" mentality.

In *concurrent* collaboration, multiple teachers are working on the same learning activity at the same time. This can range from co-creation of learning activities to in-class scenarios demonstrating collaborative modelling or even using teaching assistants to help with in-class active learning activities. These types of collaboration require typical concurrent editing capabilities, including mechanisms for conflict resolution, and collaborative awareness in modelling tools.

Teacher–student collaboration Teachers collaborate with students

(i) *synchronously* to model behaviours and help students overcome difficulties, as well as
(ii) *asynchronously* to provide feedback and assessment of modelling work.

This requires effective mechanisms for appropriate interaction. For example, it is pedagogically inappropriate for the teacher to directly change the student's model. Instead, teachers need to be able to suggest model changes, provide critique and feedback on good and bad model elements, etc. Ideally, such feedback should be directly connected to the individual elements of the model, as this allows students to easily understand the context the feedback is referring to.

This requires modelling-specific mechanisms that can be used with a wide range of modelling languages and notations. In particular, the optimal mechanisms will depend on the type of concrete syntax used. For example, for textual modelling languages, mechanisms like patch annotation (e.g. in the style offered in GitHub pull requests) can work very well for providing asynchronous model feedback.

Teacher–student collaboration on a large scale is currently facilitated through one-to-many lectures and labs. This process could be enhanced by incorporating a "modelling-bot" functionality, as discussed later in Sect. 4.4.1. Essentially, this bot will be trained using resources from teachers, past assessments, and relevant data, aiming to provide instant feedback at scale. It is important to note that while this does not replace the unique teacher–student feedback mechanism, it does automate the correction of common mistakes and repetitive aspects of the process, benefiting both students and teachers.

It is worth mentioning that while this approach is applicable across various subjects, modelling particularly stands to benefit due to the nature of its assessment. Unlike subjects with clear right or wrong answers, like mathematics, modelling involves diagrams assessed based on quality [40, 52], which requires a more nuanced evaluation that may be better supported through the interactive nature of a bot.

4.3.5 Traceability

As discussed in Sect. 3.1, when teaching model-based development, students are introduced to modelling the different aspects or views of a system at the right level of abstraction using the most appropriate formalism. It is also essential for students to understand that no model is an island. Models developed in downstream activities must conform to models created in upstream activities. Tool support is required to demonstrate the dependencies between the multitude of models, possibly with the use of megamodels [7] and/or macromodels [66].

For software engineering projects that use modelling, an MTT should provide capabilities for automatically checking if traceability is maintained between the modelling artefacts. The checker would assess conformance between models, for example, whether the design class diagram conforms to the domain model, or whether the sequence diagram conforms to the textual use cases for an application. The checker

could annotate models and generate traceability information to highlight the missing links between two modelling levels.

4.4 Support for students

Next, we discuss requirements specifically to support students.

4.4.1 Modelling assistants

In recent years, numerous techniques that assist modellers have been proposed. These recommendation systems offer suggestions to the modellers based on certain factors, e.g. similarity between the current content of the model and existing models in knowledge repositories, without specifically focussing on teaching [15, 19, 30, 50, 69].

The participants of the workshop felt that existing approaches (e.g. DoMoBot [65] and SOCIO [63]) are going in the right direction, but that more can be done with respect to teaching support. For example, chatbots and assistants for modelling should take the current level of the student into account when providing feedback, i.e. not reveal the correct solution immediately, but provide *just enough of a hint* for the student to be able to determine the solution themselves. Furthermore, modelling assistants that determine a specific weak point of a student could point them to the relevant teaching material or suggest specific training exercises. As collaborative learning is often more motivating [25], the modelling assistant could also group students according to their level and suggest collaborative exercises tailored to the group's learning needs.

4.4.2 Constraining a modelling language

Modelling languages often provide many concepts and features, some of which are fundamental, whereas others are advanced, i.e. not used very often and/or potentially difficult to grasp for newcomers. For example, whereas the concepts of class, attribute, and binary associations are fundamental to modelling using class diagrams, concepts such as association classes or n-ary associations are less often used. To reduce the cognitive load on students and to guide them in their learning, we imagine that it would be very useful if MTTs provided support for restricting the available features of a modelling language for a given exercise to those required for solving it. Over time, and as the expertise of the student grows, more difficult exercises would give access to more advanced features.

Restricting the available features of a language to a subset of the language also makes sense to tailor it to the modelling activity that is being carried out. For example, when creating a domain model, classes typically do not declare any operations, nor does one bother modelling visibility. On the other hand, when a class diagram represents a software design, then being able to define operations for classes is essential. In this case, association classes or n-ary associations are typically not used.

4.4.3 Collaboration

Modelling is often collaborative and MTTs should support this.

Direct collaborative modelling Modelling tools for teaching should provide opportunities for students to store their models on a standard version control system to allow asynchronous, offline collaboration with other students. This requires students to understand version control systems and their practical use. Additionally, it may be important to provide adapted support for model comparison and merging.

Students also expect to be able to collaborate synchronously; that is, multiple students would edit the same model at the same time, possibly each using their computer. Some modelling tools already support such collaborative modelling. For example, tools such as Magic Draw, Enterprise Architect, and MetaEdit+ [42] already support online collaboration, including across different domains [35].

Indirect collaboration Students can benefit from learning from other students' work [20]. One way of doing this is to enable peer feedback, where students comment on other students' solutions to the same assignment. For this, as mentioned in Sect. 4.3.2, it would be beneficial if feedback could be given through direct annotation of models.

Another, more implicit mechanism is for students to receive feedback and suggestions based on an analysis of other students' submissions. For example, where a student has not been able to complete a task successfully, the system could identify other students who submitted similar, but successful solutions and use the delta to suggest parts of the model the student should focus on to develop a correct solution. Similar systems have been experimented with in the context of assessing programming tasks [32], but they need to be developed for modelling education.

4.4.4 Gamification

There is strong interest in utilising gamification in modelling, yielding promising initial results [13]. However, the predominant efforts thus far have involved modellers manually creating ad hoc gamification environments tailored to specific experimental scenarios. A genuine push towards fully integrating gamification into modelling tools remains outstanding. Notable exceptions, like [13, 24, 26, 60, 75], can be regarded as early endeavours addressing the gamification of modelling, particularly in the context of modelling educa-

tion. These authors have introduced technical solutions that target specific learning objectives, such as data/process, and UML class diagrams modelling.

New MTTs should incorporate a fully fledged gamification environment. The game definition mechanisms within this framework must distinguish various learning dimensions, including learning abstraction, a modelling language, or a modelling tool.

5 Technical requirements

In this section, we list the technical requirements for our envisioned MTTs. We split the discussion into requirements to help the students, the teachers and to support the development of the tools themselves.

5.1 For students

To allow them to focus on the actual modelling activity, students should be able to use modelling tools *without the need for installation or complex plugin management*. Web-based, playground-style solutions (like the Epsilon Playground[7] or [5]) may be one approach here, but alternative options (e.g. integration into common platforms like VS Code) may also be appropriate.

Students use many different computing platforms, and it is essential that the modelling tools used in teaching are available across these platforms and do not significantly change the way they work when switching between platforms.

Platform independence especially concerns both the modelling artefacts and the user interface. Regarding the former, models cannot break or deviate in their behaviour when working on different platforms. Regarding the latter, it is important that demonstrations of teachers or teaching assistants can be directly reproduced by students, independent of, for example, their operating system.

Students have many demands on their time and will rarely be able to complete an activity in one sitting.

It is, therefore, essential that educational modelling tools provide easy ways of saving intermediary model states and continuing from such a saved state at a later time. Saving can be internal to the tool (e.g. to an internal database) or can be to an external file. Importantly, it must be possible to export work in a format that is appropriate for submission of assignments (or to submit directly from within the tool if more appropriate). This includes simple choices such as ensuring no absolute paths are used in exported files.

5.2 For teachers

Teachers can have different requirements depending on a number of factors, including the extent to which the tool will be used to support learning and assessment activities within the course and the available technical skills and infrastructure.

Teachers who plan to use the tool extensively in their course and who have technical skills and resources may like to deploy a version of the tool (e.g. through a Docker image) on local resources. This allows them to control when to upgrade to the next version of the tool, to patch/work around any identified issues with the tool, and to shield students from unexpected events. On the other hand, teachers who only plan to use the tool for a small, non-critical part of their course are likely to prefer a ready-to-use public instance (like those offered by the Epsilon Playground and UmpleOnline [49]). In this case, teachers may require configuring the tool with custom examples (e.g. metamodels, grammars, models, model management programs) to better integrate with the rest of the teaching material (e.g. example systems/domains discussed in lectures). Ideally, doing so should not require teachers to install an instance of the tool on their own resources; publicly available instances should provide built-in support for this. The Epsilon Playground, for example, allows users to configure its set of examples by specifying the location of a JSON document as a URL parameter.[8] Finally, students should be able to share their work with teachers (e.g. by generating and sharing a unique URL as is the case with the Epsilon Playground) and teachers must be able to export their students' work and persist them in a standard format to carry out further activities (e.g. inspection, marking).

5.3 For MTT developers

MTTs should demonstrate common characteristics of well-engineered community-driven software to facilitate adoption and continued development. The implementation should be accompanied by tests that demonstrate the correct behaviour of the tool and protect it against regressions. The architecture and code should be modular, to allow adopters to add/disable components and services. The code should be implemented using languages and frameworks that have a substantial user basis to facilitate long-term maintenance. Finally, the use of a permissive open-source licence is necessary to facilitate contributions by the community.

[7] https://eclipse.dev/epsilon/playground/.

[8] See https://eclipse.dev/epsilon/doc/articles/playground/#custom-examples.

6 Modelling tool examples

To illustrate the requirements discussed in Sects. 3 to 5, we present in this section how some of the existing academic tools nowadays address them.

6.1 Executable modelling with examples in Umple

Umple [49] is an open-source modelling language and technology designed to improve both education about modelling and open-source development with models. Umple's textual syntax (*cf.* Sect. 3.1.2) incorporates a variety of model types including class models, state machines, and feature models. Most graphical representations can also be edited, with the edits instantly reflected in the text. Umple generates code from all its modelling constructs, enabling model-driven design of complete systems (*cf.* Sect. 3.1.5). Where needed, user-defined code in traditional programming languages including Java, PHP, and Python is embedded in the same files as the textual models. The result is a textual format that resembles what students would be used to, given their experience with any other compiler.

To make it clear to students that serious systems can be built with models, an extensive online user manual and set of examples are available. Students also see the write-compile-execute-repeat cycle and experience satisfaction when they get a system working that is built from models.

A key feature of Umple is its extensive automatic analysis of model consistency (*cf.* Sect. 3.1.3). Several hundred error messages and warnings are produced regarding issues Umple detects in models. Each error or warning also has its own dedicated manual page, with live examples showing how to correct errors.

6.2 Constraining modelling languages in TouchCORE

TouchCORE is an academic modelling tool that focuses on model reuse following the Concern-Oriented Reuse approach. TouchCORE currently supports several standard modelling notations, such as class diagrams, sequence diagrams, and use case diagrams. It is also possible to define new domain-specific languages and augment them with concern-oriented capabilities by integrating them into TouchCORE using a plugin mechanism.

In addition to the language metamodel, a language definition in CORE requires the language designer to specify a set of *language actions* that define the construction semantics. The actions encapsulate complete editing steps that are used by the modeller when elaborating a model using the language. In other words, the language actions constitute the *API* for building models with the language.

On top of the language actions, TouchCORE provides the notion of a *perspective* [4], which allows a modeller to group a set of language actions and make them available to the user for creating models *for a specific purpose*. For example, a user can select the *Domain Modelling* perspective, which then opens the class diagram editor, but is configured in such a way that it shields the user from the full power of UML class diagrams (*cf.* Sect. 4.4.2). In the *Domain Modelling* perspective, it is not possible to create operations for classes, nor can one specify visibility for attributes or navigability of associations. On the other hand, if the user selects the *Design Modelling* perspective, then the creation of operations and specification of visibility and navigability is allowed. On the other hand, the use of association classes and n-ary associations (where $n \geq 3$) is disallowed.

6.3 Web-based playgrounds: addressing no-installation requirements, teacher collaboration, and constrained modelling activities

In the world of programming languages, web-based playgrounds have been around for some time. One of the most well-known such environments is perhaps www.w3schools.com [62], which has been offering online training on the basics of a wide range of programming languages, web frameworks, etc., for a long time.

With playgrounds no software installation or configuration is required (cf. Sect. 5.1). All learning activities are accessible from within a web browser with no need for the student to install complicated tools or set up development environments. Furthermore, playgrounds are typically set up for specific activities and provide a bespoke, typically simplified user interface exposing only the minimal functionality required to support the activity (cf. Sect. 4.4.2). As a result, students can focus on the interactions required for the activity without distractions from tool or language complexity.

As a result of these benefits, playgrounds have been experimented with by MDE tool developers, primarily as a mechanism for enriching the online documentation of their tools. For example, the Epsilon Playground provides a range of examples for using the Epsilon toolkit [44]. Similarly, the Langium Playground[9] allows experimentation with the Langium language workbench.

The MDENet education platform [5][10] generalises from these ideas and aims to develop a playground into which different MDE tools and techniques can be easily integrated. Technically derived from the Epsilon Playground, the platform introduces declarative specifications for learning activities and MDE tools. MDE tools are expected to be packaged as web services offering an API for running tool-specific actions (for example, a model-transformation

[9] https://langium.org/playground/.

[10] https://github.com/mdenet/educationplatform/.

tool would offer an API endpoint which can accept a model, metamodel, and transformation specification and return the transformed model). The platform then allows declarative activity specifications to draw on a whole range of tools available in this way and constructs a dedicated playground from each such specification. Activity specifications and the associated files can be stored in a GitHub repository, making them accessible to students, for example, via GitHub classroom and similar mechanisms. Students are even able to save back their progress via the platform (cf. Sect. 5.1), which will create a commit in the underlying repository (assuming the student has sufficient access rights).

Making activity specifications explicit as declarative models via the GitHub platform makes it possible for these to be *shared and co-developed by different teachers*, helping to address some of the requirements on teacher collaboration (cf. Sect. 4.3.4).

6.4 Teaching language engineering in graphical, textual, and projectional language workbenches

Students should be exposed to developing DSMLs using different modelling paradigms (cf. Sect. 3.2.1). AToMPM [74] is a web-based graphical modelling environment. Being bootstrapped, its editor is completely customizable, enabling teachers to adapt the tool for specific assignments. Focussing solely on graphical models, students define the concrete syntax of their DSML with SVG elements. As for the abstract syntax, students usually define it using the built-in UML class diagram; however, teachers can define other metamodelling languages (such as Entity-Relation) as well. AToMPM also enables the definition of the semantics of the DSL with rule-based model transformations. Students develop different algorithmic skills than needed when programming, being a declarative specification based on graph transformations. These transformations are mainly used to refactor, simulate, or analyse models. Using this tool, students can observe live animations of their model while running the transformation continuously or step-by-step.

The Eclipse Modelling Framework offers a variety of modelling tools that allow students to model both using a graphical or textual. However, Eclipse's strength is mostly in the former. A prominent textual modelling is Xtext [29], a textual model editor generator. Using Xtext, students learn to define grammars as well as other core language engineering components such as validators, postprocessors, and textual styling. It is compatible with Xtend [28], a template-based code generation engine. This allows students to develop code generation skills from high-abstraction models targeting different programming languages and platforms. Within the Eclipse realm, they can also use the ATL [39] model-to-model transformation engine, specifically tailored to translate models across modelling languages. ATL transformations are developed with declarative rules augmented with OCL-like expressions.

Defining DSMLs with projectional language workbenches, like MPS, differs from the previous modelling paradigms when developing the concrete syntax of the language. In MPS, students define textual projections of their language. They then define a Java code generator to give the operational meaning to their language. Alternatively, Gentleman [46] is a web-based projectional editor generator where students develop and use DSMLs in a web browser. In this case, they define projections for each metamodel element in the form of a group of HTML widgets. Gentleman also supports graphical projections. Teachers can easily integrate the generated projectional editors into full-fledged web applications demonstrating to students how MDE technologies can be interleaved with programming technologies seamlessly.

6.5 Gamification with PapyGame

As a plugin for the Papyrus modelling tool,[11] PapyGame aims to gamify the learning of modelling by integrating a game view within the Papyrus UML editor. This gamification (cf. Sect. 4.4.4) is made possible through the utilisation of a Gamification Engine, enabling the implementation of key gaming concepts associated with specific learning paths. The tool utilises a gamification design framework accessible through user interfaces in a web browser. This integration serves to enhance the learning experience for users, providing both effective and enjoyable ways to learn and practice modelling skills. For educators, PapyGame stands as a valuable asset to enrich their students' learning experiences and foster improved learning outcomes.

6.6 Skills, concepts, OER, and learning paths in MDE through the ENCORE platform

The ENCORE platform [14] has been designed and developed to support teachers and learners both in designing and in delivering personalised learning paths (cf. Sects. 4.1 and 4.2) in MDE based on a set of available OERs (cf. Sect. 4.3.1). In ENCORE, a learning path typically consists of a series of lessons and modelling activities that build on one another to create a cohesive educational experience. The process for creating a learning path generally involves two main steps: i) creating the learning path by identifying the learning objectives, selecting relevant content and assessment, and using the most appropriate instructional strategies, e.g. lectures, group activities or assignments; and ii) delivering the learning path to students while monitoring students' progress and providing support when needed. Through the use of the ENCORE Enabler for Educators (E4E), educators can access

[11] https://ci.eclipse.org/papyrus/view/PapyGame/.

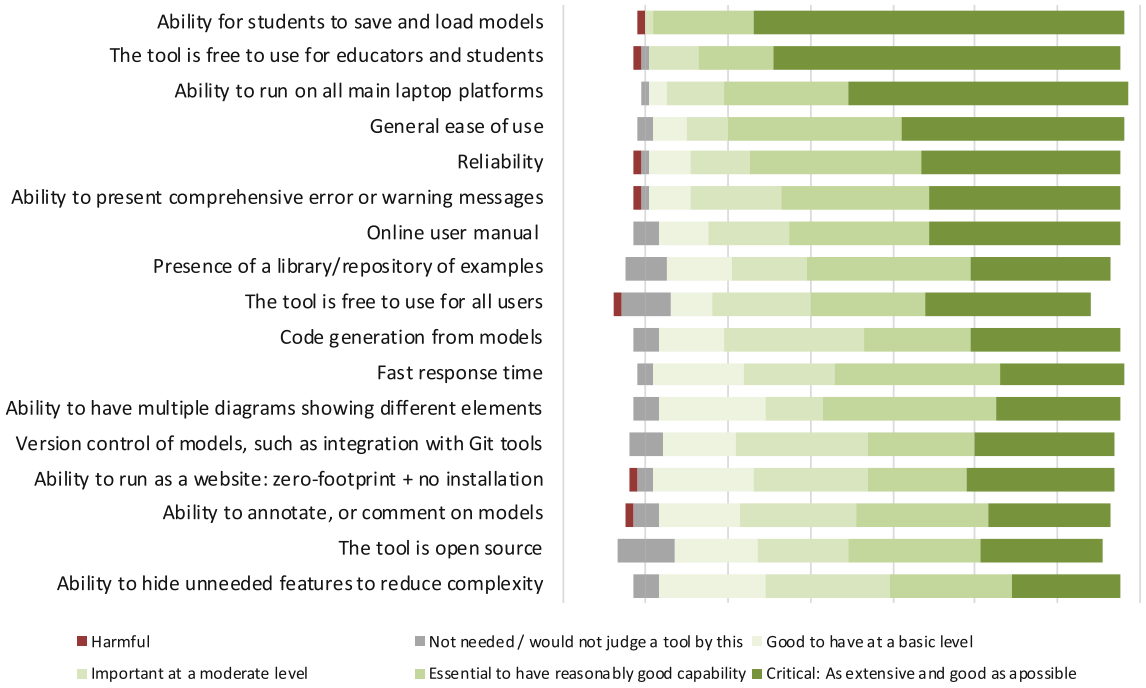


Fig. 3 Top responses to Q7 to 13 on required features of MTTs. This was a semantic-difference scale question and each coloured bar represents the percentage of responses for each possible answer, centred on "Not needed"

the ENCORE database and include in the specific learning paths the relevant OERs that target specific skills in MDE. A second enabler, the ENCORE Enabler for Learners (E4L), supports the delivery of the resulting learning path to students by leveraging notebook interfaces. Each notebook can be configured to provide and assess a specific learning path and can be augmented with gamification mechanisms to promote students' learning engagement.

7 Conclusion

While modelling is an important activity in software engineering, modelling tools are often not good enough to be used efficiently and effectively in teaching contexts. Based on discussions in a 1-day working session at MODELS 2023, we have reported a catalogue of requirements on future modelling tools for teaching (MTTs).

We conducted an international survey and asked participants which features they would value particularly highly in a modelling tool for teaching. Figure 3 gives an overview over the top 17 features based on participants' responses on a semantic-difference scale.

Notably, participants strongly care about the **usability** aspects of modelling tools. Respondents consistently underscore the significance of features such as *general ease of use*, *fast response time*, and the *ability to hide unneeded features*. These elements are foundational to the user experience, as they directly impact **efficiency** and **user satisfaction**. A tool's intuitive interface and seamless navigation are pivotal in facilitating the modelling process, enabling users to focus on their tasks without grappling with unnecessary complexities. Note, also, that usability already played an important role in the responses to the initial survey we conducted directly during the workshop.

Moreover, participants emphasise **reliability**. This encompasses not only the stability and robustness of the tool but also its ability to consistently deliver accurate results and perform as expected under various conditions. Users prioritise tools that they can rely on to execute tasks reliably, minimising the risk of errors and disruptions in their workflow.

In addition to usability and reliability, the data highlights the importance of functionality geared towards **enhancing collaboration** and **learning**. Features such as the *ability for students to save and load models*, *version control*, and the presence of a *library/repository of examples* facilitate knowledge sharing, iteration, and experimentation.

Furthermore, **accessibility** and **affordability** emerge as significant determinants of tool adoption and usage. The ability to *run on all main laptop platforms* and the availability of *free usage for educators and students* are important to respondents. These features ensure inclusiveness and democratise access to modelling tools, making them accessible to a broader audience regardless of their technological or financial constraints.

Developing the next generation of MTTs must be a community effort. We invite everyone interested in these topics

to join our efforts and build better tools for future teaching of modelling.

Acknowledgements We thank all the participants of the Workshop on Modelling Tools for Teaching (https://modellingtoolsforteaching.github.io/) held in conjunction with MODELS 2023 in Västeras, Sweden for their contributions to the discussions that led to this paper. Zschaler and Barnett's contributions were partially supported by the UK Engineering and Physical Sciences Research Council (EPSRC) as part of MDENet—the expert network in model-driven engineering—(Grant Reference EP/T030747/1). Abrahão's contributions were partially supported by the State Research Agency (AEI) under the UCI-Adapt Project (PID2022-140106NB-I00). Kienzle's contributions were partially supported by Junta de Andalucía under Project QUAL21 010UMA.

References

1. Abrahão, S., Bordeleau, F., Cheng, B.H.C., Kokaly, S., Paige, R.F., Störrle, H., Whittle, J.: User experience for model-driven engineering: challenges and future directions. In: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, Austin, TX, USA, September 17–22, 2017, pp. 229–236. IEEE Computer Society (2017)
2. Abrahão, S., Gravino, C., Insfrán, E., Scanniello, G., Tortora, G.: Assessing the effectiveness of sequence diagrams in the comprehension of functional requirements: results from a family of five experiments. IEEE Trans. Softw. Eng. **39**(3), 327–342 (2013)
3. Alario-Hoyos, C., Bote-Lorenzo, M.L., Gómez-Sánchez, E., Asensio-Pérez, J.I., Vega-Gorgojo, G., Ruiz-Calleja, A.: Glue! An architecture for the integration of external tools in virtual learning environments. Comput. Educ. **60**(1), 122–137 (2013)
4. Ali, H., Mussbacher, G., Kienzle, J.: Perspectives to promote modularity, reusability, and consistency in multi-language systems. Innov. Syst. Softw. Eng. **18**(1), 5–37 (2022)
5. Barnett, W., Zschaler, S., Boronat, A., Garcia-Dominguez, A., Kolovos, D.: An online education platform for teaching MDE. In: Proceedings of Educators Symposium at MODELS 2023 (2023)
6. Bezivin, J., France, R., Gogolla, M., Haugen, O., Taentzer, G., Varro, D.: Teaching modeling: why, when, what? In: Ghosh, S. (ed.) Models in Software Engineering, pp. 55–62. Springer, Berlin (2010)
7. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the large and modeling in the small. In: European Workshop on Model Driven Architecture, pp. 33–46. Springer (2003)
8. Bian, W., Alam, O., Kienzle, J.: Is automated grading of models effective? Assessing automated grading of class diagrams. In: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, pp. 365–376 (2020)
9. Bousse, E., Degueule, T., Vojtisek, D., Mayerhofer, T., Deantoni, J., Combemale, B.: Execution framework of the GEMOC Studio (tool demo). In: International Conference on Software Language Engineering, pp. 84–89. Association for Computing Machinery (2016)
10. Bruneliere, H., Burger, E., Cabot, J., Wimmer, M.: A feature-based survey of model view approaches. Softw. Syst. Model. **18**, 1931–1952 (2019)
11. Bruneliere, H., Cabot, J., Dupé, G., Madiot, F.: Modisco: a model driven reverse engineering framework. Inf. Softw. Technol. **56**(8), 1012–1032 (2014)
12. Bucchiarone, A., Cabot, J., Paige, R.F., et al.: Grand challenges in model-driven engineering: an analysis of the state of the research. Softw. Syst. Model. **19**, 5–13 (2020)
13. Bucchiarone, A., Savary-Leblanc, M., Le Pallec, X., Cicchetti, A., Gérard, S., Bassanelli, S., Gini, F., Marconi, A.: Gamifying model-based engineering: the PapyGame experience. Softw. Syst. Model. **22**(4), 1369–1389 (2023)
14. Bucchiarone, A., Vazquez-Ingelmo, A., Garcia-Holgado, A., Barandoni, S., Schiavo, G., Mosser, S., Pierantonio, A., Zschaler, S., Barnett, W.: Towards personalized learning paths to empower competency development in model driven engineering through the ENCORE platform. In: Educators Symposium at MODELS 2023 (2023)
15. Burgueño, L., Clarisó, R., Gérard, S., Li, S., Cabot, J.: An NLP-based architecture for the autocompletion of partial domain models. In: La Rosa, M., Sadiq, S., Teniente, E. (eds.) Advanced Information Systems Engineering, pp. 91–106. Springer, Cham (2021)
16. Camp, T., Adrion, W.R., Bizot, B., Davidson, S., Hall, M., Hambrusch, S., Walker, E., Zweben, S.: Generation cs: the growth of computer science. ACM Inroads **8**(2), 44–50 (2017)
17. Castañeda, L., Selwyn, N.: More than tools? Making sense of the ongoing digitizations of higher education. Int. J. Educ. Technol. High. Educ. **15**(1), 1–10 (2018)
18. CC2020 Task Force: Computing Curricula 2020: Paradigms for Global Computing Education. Association for Computing Machinery, New York (2020)
19. Cerqueira, T.G.O., Ramalho, F., Marinho, L.B.: A content-based approach for recommending UML sequence diagrams. In: Gou, J. (ed.) The 28th International Conference on Software Engineering and Knowledge Engineering, SEKE 2016, Redwood City, San Francisco Bay, USA, July 1–3, 2016, pp. 644–649. KSI Research Inc. and Knowledge Systems Institute Graduate School (2016)
20. Chakarvarti, P.: Investigating the effectiveness of peer feedback in developing critical thinking skills in undergraduate students. J. Educ. Rev. Provis. **2**(3), 91–95 (2022)
21. Cicchetti, A., Ciccozzi, F., Pierantonio, A.: Multi-view approaches for software and system modelling: a systematic literature review. Softw. Syst. Model. **18**, 3207–3233 (2019)
22. Ciccozzi, F., Famelis, M., Kappel, G., Lambers, L., Mosser, S., Paige, R.F., Pierantonio, A., Rensink, A., Salay, R., Taentzer, G., Vallecillo, A., Wimmer, M.: 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 (2018)
23. Combemale, B., Crégut, X., Dieumegard, A., Pantel, M., Zalila, F.: Teaching mde through the formal verification of process models. Electron. Commun. EASST **52** (2012)
24. Cosentino, V., Gérard, S., Cabot, J.: A model-based approach to gamify the learning of modeling. In: Proceedings of the 5th Symposium on Conceptual Modeling Education and the 2nd International iStar Teaching Workshop Co-located with the 36th International Conference on Conceptual Modeling (ER 2017), Valencia, Spain, November 6–9, 2017, pp. 15–24 (2017)

25. Damayanti, E., Nur, F., Anggereni, S., Taufiq, A.U.: The effect of cooperative learning on learning motivation: a meta-analysis. Buletin Psikologi **31**(1), 116 (2023)
26. De Smedt, J., De Weerdt, J., Serral, E., Vanthienen, J.: Gamification of declarative process models for learning and model verification. In: Reichert, M., Reijers, H.A. (eds.) Business Process Management Workshops, pp. 432–443. Springer, Cham (2016)
27. Devore-McDonald, B., Berger, E.D.: Mossad: Defeating software plagiarism detection. Proc. ACM Program. Lang. **4**(OOPSLA), 1–28 (2020)
28. Efftinge, S., Köhnlein, J., Zarnekow, S.: Xtend—modernized java. https://eclipse.dev/Xtext/xtend/. Last visited May 10 2024
29. Efftinge, S., Köhnlein, J., Zarnekow, S.: Xtext language development framework. https://eclipse.dev/Xtext/. Last visited May 10 2024
30. Elkamel, A., Gzara, M., Ben-Abdallah, H.: An UML class recommender system for software design. In: 2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA), pp. 1–8 (2016)
31. Erdweg, S., van der Storm, T., Völter, M., Tratt, L., Bosman, R., Cook, W.R., Gerritsen, A., Hulshout, A., Kelly, S., Loh, A., Konat, G.D.P., Molina, P.J., Palatnik, M., Pohjonen, R., Schindler, E., Schindler, K., Solmi, R., Vergu, V.A., Visser, E., van der Vlist, K., Wachsmuth, G., van der Woning, J.: Evaluating and comparing language workbenches: existing results and benchmarks for the future. Comput. Lang. Syst. Struct. **44**, 24–47 (2015)
32. Gross, S., Mokbel, B., Paassen, B., Hammer, B., Pinkwart, N.: Example-based feedback provision using structured solution spaces. Int. J. Learn. Technol. **9**(3), 248–280 (2014)
33. Gómez-Abajo, P., Guerra, E., Lara, J.: Automated generation and correction of diagram-based exercises for Moodle. Comput. Appl. Eng. Educ. **31**, 08 (2023)
34. Hassane, O., Mustafiz, S., Khendek, F., Toeroe, M.: MAPLE-T: a tool for process enactment with traceability support. In: ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), pp. 759–763 (2019)
35. Herac, E., Assunção, W.K.G., Marchezan, L., Haas, R., Egyed, A.: A flexible operation-based infrastructure for collaborative model-driven engineering. J. Obj. Technol. **22**(2), 2:1-2:14 (2023)
36. Hilton, J.: Open educational resources, student efficacy, and user perceptions: a synthesis of research published between 2015 and 2018. Educ. Technol. Res. Dev. **68**(3), 853–876 (2020)
37. Holzmann, G.J.: The model checker spin. IEEE Trans. Softw. Eng. **23**(5), 279–295 (1997)
38. Hosseinibaghdadabadi, M., Alam, O., Almerge, N., Kienzle, J.: Automated grading of use cases. In: Proceedings of the 26th International Conference on Model Driven Engineering Languages and Systems, MODELS '23, pp. 106–116, New York, NY, USA. Association for Computing Machinery (2023)
39. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. Sci. Comput. Program. **72**(1), 31–39 (2008)
40. Karasneh, B., Stikkolorum, D., Larios, E., Chaudron, M.: Quality assessment of UML class diagrams. In: Proceedings of Educators' Symp at MoDELS (2015)
41. Kecskemeti, G., Casale, G., Jha, D.N., Lyon, J., Ranjan, R.: Modelling and simulation challenges in internet of things. IEEE Cloud Comput **4**(1), 62–69 (2017)
42. Kelly, S.: Collaborative modelling with version control. In: Federation of International Conferences on Software Technologies: Applications and Foundations, pp. 20–29. Springer (2017)
43. Klare, H., Kramer, M.E., Langhammer, M., Werle, D., Burger, E., Reussner, R.: Enabling consistency in view-based system development: the vitruvius approach. J. Syst. Softw. **171**, 110815 (2021)
44. Kolovos, D., Paige, R., Rose, L., Polack, F.: The Epsilon Book. http://www.eclipse.org/gmt/epsilon/doc/book/ (2009)
45. Kolovos, D.S., García-Domínguez, A., Rose, L.M., Paige, R.F.: Eugenia: towards disciplined and automated development of GMF-based graphical model editors. Softw. Syst. Model. **16**, 229–255 (2017)
46. Lafontant, L.-E., Syriani, E.: Gentleman: a light-weight web-based projectional editor generator. In: Model Driven Engineering Languages and Systems: Companion Proceedings, pp. 1–5. ACM (2020)
47. Larsen, K.G., Pettersson, P., Yi, W.: Uppaal in a nutshell. Int. J. Softw. Tools Technol. Transf. **1**, 134–152 (1997)
48. Le, T., Carbone, A., Sheard, J., Schuhmacher, M., de Raath, M., Johnson, C.: Educating computer programming students about plagiarism through use of a code similarity detection tool. In: 2013 Learning and Teaching in Computing and Engineering, pp. 98–105. IEEE (2013)
49. Lethbridge, T.C., Forward, A., Badreddin, O., Brestovansky, D., Garzon, M., Aljamaan, H., Eid, S., Orabi, A.H., Orabi, M.H., Abdelzad, V., Adesina, O., Alghamdi, A., Algablan, A., Zakaripour, A.: Umple: model-driven development for open source and education. Sci. Comput. Program. **208**, 102665 (2021)
50. Lucrédio, D., Fortes, R.P.M., Whittle, J.: MOOGLE: a metamodel-based model search engine. Softw. Syst. Model. **11**(2), 183–208 (2012)
51. Martínez, S., Wimmer, M., Cabot, J.: Efficient plagiarism detection for software modeling assignments. Comput. Sci. Educ. **30**(2), 187–215 (2020)
52. Modi, S., Taher, H.A., Mahmud, H.: A tool to automate student UML diagram evaluation. Acad. J. Nawroz Univ. **10**(2), 189–198 (2021)
53. Mustafiz, S., Vangheluwe, H.: Explicit modelling of statechart simulation environments. In: Proceedings of the 2013 Summer Computer Simulation Conference, SCSC'13, Vista, CA. Society for Modeling & Simulation International (2013)
54. Nainan, M., Balakrishnan, B.: Design and evaluation of worked examples for teaching and learning introductory programming at tertiary level. Malays. Online J. Educ. Technol. **7**, 30–44 (2019)
55. Novak, M., Joy, M., Kermek, D.: Source-code similarity detection and detection tools used in academia: a systematic review. ACM Trans. Comput. Educ. **19**(3), 1–37 (2019)
56. Object Management Group. Object constraint language. https://www.omg.org/spec/OCL, 2014. Last accessed 15 February 2024
57. Otto, D.: Adoption and diffusion of open educational resources (OER) in education: a meta-analysis of 25 OER-projects. Int. Rev. Res. Open Distrib. Learn. **20**(5), 122–140 (2019)
58. Panach, J.I., Pastor, Ó.: A practical experience of how to teach model-driven development to manual programming students. Enterp. Model. Inf. Syst. Architect. **18**(6), 1 (2023)
59. Park, C.: In other (people's) words: plagiarism by university students-literature and lessons. Assess. Eval. High. Educ. **28**(5), 471–488 (2003)
60. Pflanzl, N.: Gameful business process modeling. In: Mendling, J., Rinderle-Ma, S. (eds.) Proceedings of the 7th International Workshop on Enterprise Modeling and Information Systems Architectures, EMISA 2016, volume 1701 of CEUR Workshop Proceedings, pp. 17–20. CEUR-WS.org (2016)
61. Plante, F.: Introducing the GMF runtime. https://www.eclipse.org/articles/Article-Introducing-GMF/article.html, 2006. Last accessed 15 February 2024
62. Refsnes Data. W3Schools Website. http://www.w3schools.com. Last accessed 15 May 2024
63. Ren, R., Castro, J.W., Santos, A., Dieste, O., Acuña, S.T.: Using the SOCIO chatbot for UML modelling: a family of experiments. IEEE Trans. Softw. Eng. **49**(1), 364–383 (2022)
64. Richters, M., Gogolla, M.: Validating UML models and OCL constraints. In: Evans, A., Kent, S., Selic, B. (eds.) UML 2000–The Unified Modeling Language, pp. 265–277. Springer, Berlin (2000)

65. Saini, R., Mussbacher, G., Guo, J.L.C., Kienzle, J.: Automated, interactive, and traceable domain modelling empowered by artificial intelligence. Softw. Syst. Model. **21**(3), 1015–1045 (2022)
66. Salay, R., Mylopoulos, J., Easterbrook, S.: Using macromodels to manage collections of related models. In: Advanced Information Systems Engineering: 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8–12, 2009. Proceedings 21, pp. 141–155. Springer (2009)
67. Sağlam, T., Schmid, L., Hahner, S., Burger, E.: How students plagiarize modeling assignments. In: Proceedings of the 26th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, MODELS '23, New York, NY, USA. Association for Computing Machinery (2023)
68. Scanniello, G., Gravino, C., Genero, M., Cruz-Lemus, J.A., Tortora, G., Risi, M., Dodero, G.: Do software models based on the UML aid in source-code comprehensibility? Aggregating evidence from 12 controlled experiments. Empir. Softw. Eng. **23**(5), 2695–2733 (2018)
69. Segura, A.M., Pescador, A., de Lara, J., Wimmer, M.: An extensible meta-modelling assistant. In: 2016 IEEE 20th International Enterprise Distributed Object Computing Conference (EDOC), pp. 1–10 (2016)
70. Sønderlund, A.L., Hughes, E., Smith, J.: The efficacy of learning analytics interventions in higher education: a systematic review. Br. J. Educ. Technol. **50**(5), 2594–2618 (2019)
71. Stahl, T., Voelter, M., Czarnecki, K.: Model-Driven Software Development: Technology, Engineering. Management. Wiley, Hoboken (2006)
72. Steinberg, D., Budinsky, F., Merks, E., Paternostro, M.: EMF: Eclipse Modeling Framework. Pearson Education, London (2008)
73. Syriani, E., Luhunu, L., Sahraoui, H.: Systematic mapping study of template-based code generation. Comput. Lang. Syst. Struct. **52**(1), 43–62 (2018)
74. Syriani, E., Vangheluwe, H., Mannadiar, R., Hansen, C., Van Mierlo, S., Ergin, H.: AToMPM: a web-based modeling environment. In: MODELS'13 Invited Talks, Demonstration Session, Poster Session, and ACM Student Research Competition, volume 1115, pp. 21–25, Miami FL. CEUR-WS.org (2013)
75. Tantan, O.C., Lang, D., Boughzala, I.: Towards gamification of the data modeling learning. In: MCIS 2017: 11th Mediterranean Conference on Information Systems, Sep 2017, Genova, Italy (2017)
76. Unesco. Recommendation on open educational resources (OER). https://www.unesco.org/en/legal-affairs/recommendation-open-educational-resources-oer, November 2019. Accessed: 2023-07-19
77. Varró, D.: Automated formal verification of visual modeling languages by model checking. Softw. Syst. Model. **3**, 85–113 (2004)
78. Viyović, V., Maksimović, M., Perisić, B.: Sirius: a rapid development of DSM graphical editor. In: IEEE 18th International Conference on Intelligent Engineering Systems (INES'14), pp. 233–238 (2014)

Jörg Kienzle is a researcher at ITIS Software, Universidad de Málaga, Málaga, Spain, and Full Professor at McGill University, Montréal, Québec, Canada, where he leads the Software Composition and Reuse lab (SCORE). His research interests include model-driven software development, software product lines, separation of concerns, reuse, software composition, and modularity. Further information about him can be found at https://djeminy.github.io.

Steffen Zschaler is a Reader in Software Engineering at King's College London. His research interests are in model-driven engineering, focussing on modularity, optimisation, and the principled engineering of simulations. He is the director of MDENet, the expert network in model-driven engineering, where he has led on the development of the MDENet Education Platform to improve the accessibility of MDE tools to students. More details about his research and teaching can be found at www.steffen-zschaler.de.

William Barnett is former Research Software Engineer at King's College London who contributed to the development and maintenance of the MDENet Education Platform. His main areas of interest are model-driven development, software architecture, and embedded systems. More details about his research can be found at www.wdbar.net.

Timur Sağlam is a doctoral researcher at the KASTEL Institute at Karlsruhe Institute of Technology (KIT). He is the project lead of the widely used open-source plagiarism detector JPlag. His research interests involve software plagiarism detection, particularly obfuscation attacks on software plagiarism detectors. More details about his research and teaching can be found at https://dsis.kastel.kit.edu/staff_saglam.php.

Antonio Bucchiarone is a Senior Researcher at the Motivational Digital Systems (MoDiS) unit of the Bruno Kessler Foundation (FBK) in Trento, Italy. His research activity is focussed on many aspects of Software Engineering for Adaptive Socio-Technical Systems. He has investigated advanced methodologies and techniques supporting the definition and development of gameful systems in different domains (i.e. education, sustainable mobility, etc.) where being adaptable is a key intrinsic characteristic. He was the General Chair of the 12th IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2018). Dr. Bucchiarone is an Associate Editor of the IEEE Transactions on Intelligent Transportation Systems (T-ITS) Journal, of the IEEE Software Magazine and of the IEEE Technology and Society Magazine.

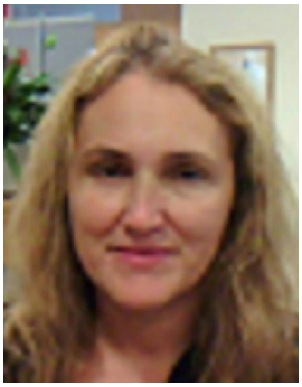
Silvia Abrahão is Full Professor of Software Engineering in the Department of Computer Science at the Universitat Politècnica de Valéncia (Spain). She is also Director of the Master's Program in Software Systems Engineering and Technology at UPV. She is associate editor for IEEE Software, ACM Transactions on Software Engineering and Methodology and Automated Software Engineering journals. She is on the editorial board of the journal Software and Systems Modelling and on the advisory editorial board of the Journal of Software: Practice and Experience. She is also a member of the ACMs Women in Computing leadership team. Her research interests include the adaptation of user interfaces using AI techniques, human factors in software engineering, quality assurance in model-driven engineering, the empirical assessment of software modelling approaches and the integration of usability/UX into software development. Contact her at https://sabrahao.wixsite.com/dsic-upv.

Eugene Syriani is a Full Professor in computer science at the University of Montreal. His main research interests fall in software engineering based on the model-driven engineering (MDE) approach, domain-specific languages, code generation, collaborative modelling, simulation-based design, digital twins, and user experience. He has a long history of teaching MDE and software design. He has also led the development of several MDE frameworks and tools. More details about his research and teaching can be found at http://www.iro.umontreal.ca/~syriani/.

Dimitris Kolovos is a Professor of Software Engineering in the Department of Computer Science at the University of York, where he researches and teaches automated and model-driven software engineering. He is also an Eclipse Foundation committer, leading the development of the open-source Epsilon model-driven software engineering platform, and an editor of the Software and Systems Modelling journal. He has co-authored more than 150 peer-reviewed papers and his research has been supported by the European Commission, UK's Engineering and Physical Sciences Research Council (EPSRC), InnovateUK and by companies such as Rolls-Royce and IBM.

Timothy Lethbridge is a Professor at the University of Ottawa, Canada. His research currently focuses on software modelling tools, particularly the user experience of such tools, their educational use, code generation, and the Umple technology. He is a licenced Professional Engineer, a senior member of both ACM and IEEE and a fellow of the Canadian Information Processing Society (CIPS). He received the IEEE Computer Society TCSE Outstanding Educator Award in 2016. He is co-general-chair of the International Conference on Software Engineering (ICSE) 2025 in Ottawa. More details about his research and teaching can be found at www.umple.org/tcl.

Sadaf Mustafiz is an Assistant Professor in the Department of Computer Science at Toronto Metropolitan University. She received her Ph.D. and M.Sc. in Computer Science from McGill University. Her research interest is in the area of model-driven software engineering with a focus on requirements engineering, domain-specific modelling, multi-paradigm modelling, process modelling, and model-based simulation. More details about her research and teaching can be found at https://cs.torontomu.ca/~sadaf/.



Sofia Meacham is a Principal Academic in Software Engineering in the Computing and Informatics Department at Bournemouth University, where she teaches model-based design and systems engineering. She has worked on several EU-funded projects in R&D, telecommunication industries and academia, and has a background in embedded systems and formal methods. Her research interests lie in domain-specific languages and model-driven engineering for diverse domains from telecommunications industries to medical applications.