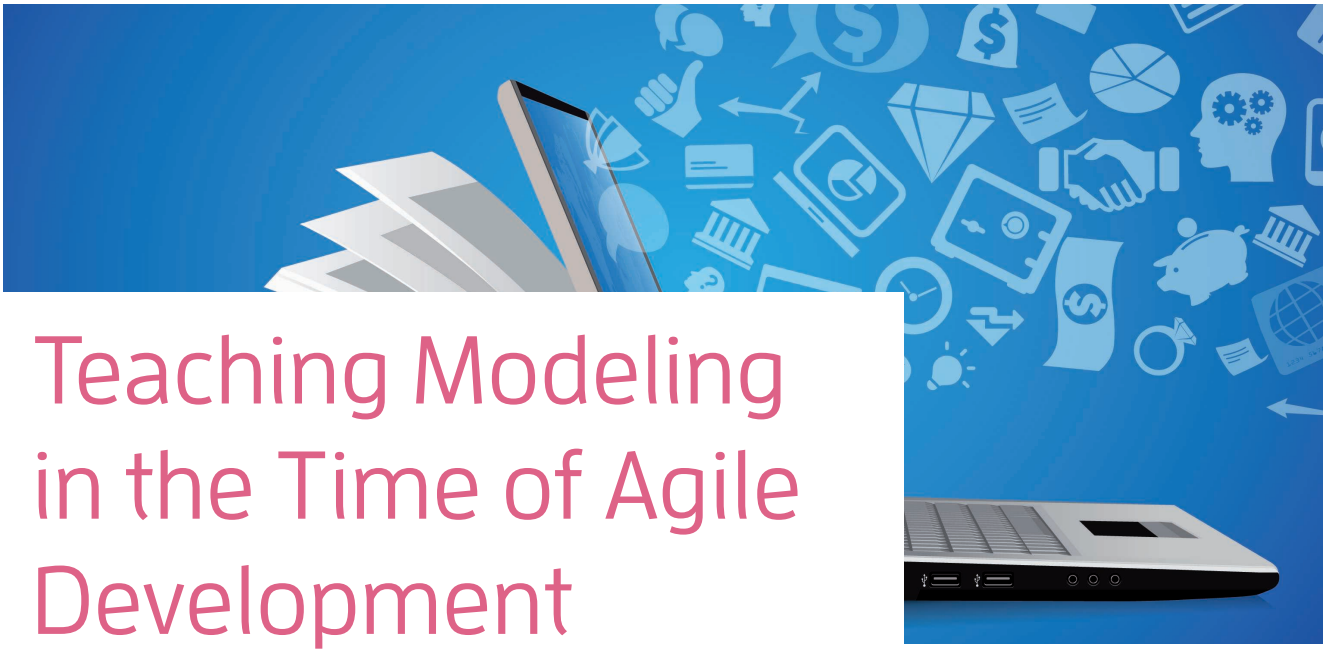
EDITOR **IRENA BOJANOVA**
NIST; irena.bojanova@computer.org



Teaching Modeling in the Time of Agile Development

Oscar Pastor, Polytechnic University of Valencia

Alfonso Pierantonio, University of L'Aquila

Gustavo Rossi, La Plata National University and National Scientific and Technical Research Council

We discuss modeling in the context of agile software development and reflect on how we, as educators, can use modeling to help improve agile practices.

Engineers (in particular, software engineers) have been motivated to find a way to confront the intrinsic and increasing complexity of their endeavors. In this sense, models have been used for centuries to abstract different aspects of a system under construction. As pointed out in Gogolla and Selic[6] and Ludewig,[8] models can describe certain aspects of a system and fundamentally act as understanding and communication means (descriptive models); they can be used to analyze and predict system properties (predictive models), and they can be employed as an implementation specification (prescriptive models). A good summary of different definitions can be found in Muller et al.[10]

In this context, model-driven engineering (MDE)[9] emerged as a discipline within software engineering, which considers models first-class citizens throughout the software process and aims to derive running applications directly and automatically from models (in general, by using transformations). MDE evolved rapidly during the 2000s, and it is considered key to success in many domains, such as railway systems, automotive applications, business process engineering, and embedded systems.[1] It simplifies software construction since developers can abstract from most technological decisions (for example, interface devices) and implementation aspects and focus on domain details. It has been shown that MDE helps improve productivity[9] and code quality.[2] However, the state of practice suggests that models are still far from being considered essential software artifacts.

Abstraction practice is strictly required for managing complexity and developing correct software.[18] Notably, putting abstraction in practice in software engineering terms means modeling. Since modeling is an essential skill for software developers, the development, manipulation, management, and comprehension of models is a

Digital Object Identifier 10.1109/MC.2022.3144929
Date of current version: 3 June 2022

relevant learning objective.[7] Therefore, educators have addressed this problem for years. In this article, we elaborate on this issue in the context of agile software development. After some brief comments on the state of practice in modeling and the way we teach about models, we reflect on how we, as educators, can help improve current agile practices.

STATE OF PRACTICE

The history of software engineering shows how abstraction is the fundamental development notion: from machine code and assemblers to programming language support, the level of abstraction is always evolving from a lower one to a higher one. Models should be the natural next step. As happens in other engineering disciplines, more abstraction should lead to better software production methods, making software engineering an accurate process. Surprisingly, this is not the case. Software engineering is too frequently perceived as closer to a (technological) handicraft-centered activity, strongly dependent on skilled programmers, not expert modelers.

As early as 1971, Teichroew and Sayani[16] stated that "the size, importance and cost of systems building provide an opportunity for the investigation of ways to improve the (software production) process." More than 50 years later, programming environments have constantly progressed, but current practice for design, programming, and testing activities still relies on substantial manual effort. Models should facilitate the automation of the systems building process. Why is that goal not being achieved? Is it unreachable? Is it worthless, assuming that conventional (not model-driven) software production is enough? If we talk about improving abstraction to better conceptualize and accurately represent reality in a software application, shouldn't a sound use of modeling be independent of what software development process is selected? These are the questions we discuss in this work.

Many authors have surveyed the extent to which modeling and MDE are used in industry (see, for example, Gorschek et al.,[12] Heldal et al.,[13] and Hutchinson et al.[14]). Specifically, Heldal et al.[13] analyze when descriptive and prescriptive models are used. While there is certainly a niche for MDE (in Heldal et al.,[13] the survey covers huge companies working on embedded systems), the mainstream use of agile approaches (which emerged as a counterpart of monolithic ones) did not come with a similar prevalence of the use of models (even descriptive ones). This mismatch is strongly reflected in developers, as discussed in Gorschek et al.[12] The reasons might have originated in corporate practices, such as discouraging the use of models because of an extreme interpretation of one of agile's principles: "Value working software over comprehensive documentation." This interpretation has also hampered the introduction of user-centered development approaches in the agile universe.[15] However, as discussed in the following, it might also be a product of problems in the education of developers.

> Engineers (in particular, software engineers) have been motivated to find a way to confront the intrinsic and increasing complexity of their endeavors.

MODELING AND MDE EDUCATION

We can get a good idea of why, what, and how we teach about modeling by reading the proceedings of educator symposiums held with the Association for Computing Machinery/IEEE International Conference on Model-Driven Engineering Languages and Systems (MODELS) (see, for example, the MODELS 2021 Educators Symposium webpage[11]). Specifically, from Kuzniarz and Börstler[7] we learn that we must include modeling in the curriculum "to encourage and stimulate thinking at high abstraction levels ... to enable and ensure successful development of software" and "to be competitive in the labor market." Ciccozzi et al.[5] survey the way we teach modeling and MDE. Besides interesting findings about course content and tools, negative aspects were explored. From our point of view, two of them are remarkable: 1) the lack of maturity of existing tools and 2) students having difficulty understanding abstraction and conceiving modeling as quite different from programming. While we, as educators, have almost no control over the former, we should improve our practices to deal with the latter, especially considering how it might impact industry practices.

APPROACHES AND IDEAS

Agile development intends to improve the software production process, which is necessarily linked to getting better abstraction capabilities. Models provide the right answer. Beyond using models for communication and to facilitate understanding, they should be the key artifact that guides software development, providing as many automation facilities as possible to connect abstract descriptions with their associated software representations. As suggested in Bucchiarone et al.,[4] we believe that the agile development wave provides an opportunity to revisit the model-driven main goal, facilitating the design of a software production process where enterprise models and software applications are conceptually aligned through the construction of the right models and

transformations. Given that models are essential to conceptualize components that must be represented in a software system, modeling correctly and building the right ones should become the most important software engineering activity, which is naturally related to teaching.

Modeling properly implies training software engineers in abstracting and conceptualizing correctly. Building the right models requires making software engineers become aware of the languages that apply to different abstraction levels [for example, Business Process Modeling Notation for business process models, i* for goal-oriented requirements models, Unified Modeling Language (UML) class diagrams for system structure models, and so on]. One of the main inhibitors of modeling in practice is the lack of a well-established "teaching modeling" body of knowledge, a gap that is only partly filled by the work in Burgueño et al.[3] Modeling is about abstracting, and how to teach and assess how good a student is at it is not simple. It means evaluating how well he or she conceptualizes, which requires skills and abilities to grasp insights and knowledge blurred in the intricacy of the application domain.

This modeling dimension should be on top of programming as an essential topic in software engineering teaching. Some problems that need to be precisely solved to achieve this objective include the following:

1. *Foundations*: We need these to have a universal, widely accepted and used definition of what a model is, providing a precise definition that is ontologically well grounded.
2. *Better tooling*: This is important to reinforce efficient, adequate, flexible, usable, and reliable (in other words, mature) tool support, facilitating the use of models in software production and making it feasible to use conceptual programming-based tools, where models go beyond a merely communicational dimension, becoming a trustworthy software artifact (as discussed in Embley et al.[17]).
3. *Revised syllabi*: These would enable us to assess and rethink how we teach abstraction and modeling. Most syllabi treat these subjects in a perhaps unrelated way. Consequently, the assimilation of abstraction (and its practice) is not consistently pursued and, to a certain degree, depends on students' attitudes and curiosity. One possibility is to use early courses on object orientation to introduce modeling instead of (only) programming. Also, as suggested in Ciccozzi et al.,[5] we should try to use project-oriented and hands-on learning. The availability of tools that permit round-tripping between code and models [for example, Visual Paradigm (https://www.visual-paradigm.com)] would make students perceive models as a functional part of their projects alongside the code and not a way to procrastinate what they are mainly interested in: coding, coding, and coding! As mentioned, the lack of educational modeling tools with reduced accidental complexity should also be addressed since tooling is another critical issue.

> Beyond using models for communication and to facilitate understanding, they should be the key artifact that guides software development.

Emphasizing the relevance of modeling in software engineering teaching must consider that modeling abilities among students should play a crucial role. Correctly abstracting a real system that is represented in a computer requires advanced conceptualization skills, which is not easy to convey and evaluate. Some students seem readier to do it well than others, but in any case, a software engineering student should not get a degree without possessing a solid modeling ability. Such difficulties can be mitigated by replanning the way abstraction is taught.

It is not just a student issue. Modeling is hard to teach. Good practices should be widely accepted and employed by educators. Assessing the syntax and semantic quality of models should become a task supported by a sound ontological commitment. Even delimiting the set of rules that a (simple) UML class diagram should follow (as a kind of correction guide) remains an open question (for example, simple aspects, such as whether classes without attributes and associations with the same names should be allowed, do not have precise, definitive answers). This represents a significant issue, as there is broad diversity among modeling languages.

From a more "tactical" point of view, we can profit from the popularity of low-code platforms to demonstrate how models and model-driven development (usually "hidden" behind visual editors, as indicated in Bucchiarone at al.[1]) improve the productivity, quality, and cost effectiveness of software development. Finally, we must prove how informal modeling (sketches, wireframes, and so on) can be easily accommodated in the agile cycle to improve communication, understanding, and agreement, as suggested in Bucchiarone at al.[1] Whether this will be a successful strategy is not easy to say.

What is worth remarking is that what makes a programmer a good software engineer is the ability to use abstraction fruitfully—and the state of the art in software abstraction is MDE.

REFERENCES

1. A. Bucchiarone *et al.*, "What is the future of modeling," *IEEE Softw.*, vol. 38, no. 2, pp. 119–127, 2021, doi: 10.1109/MS.2020.3041522.
2. J. I. Panach *et al.*, "Evaluating model-driven development claims with respect to quality: A family of experiments," *IEEE Trans. Softw. Eng.*, vol. 47, no. 1, pp. 130–145, 2021, doi: 10.1109/TSE.2018.2884706.
3. L. Burgueño *et al.*, "Contents for a model-based software engineering body of knowledge," *Softw. Syst. Model.*, vol. 18, no. 6, pp. 3193–3205, 2019, doi: 10.1007/s10270-019-00746-9.
4. A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: An analysis of the state of the research," *Softw. Syst. Model.*, vol. 19, no. 1, pp. 5–13, 2020, doi: 10.1007/s10270-019-00773-6.
5. F. Ciccozzi *et al.*, "How do we teach modeling and model-driven engineering? A survey," in *Proc. 21st ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst., Companion*, 2018, pp. 122–129, doi: 10.1145/3270112.3270129.
6. M. Gogolla and B. Selic, "On teaching descriptive and prescriptive modeling," in *Proc. 23rd ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst., Companion*, 2020, pp. 1–9, doi: 10.1145/3417990.3418744.
7. L. Kuzniarz and J. Börstler, "Teaching modeling: An initial classification of related issues," in *Proc. Electron. Commun. EASST 7th Educator's Symp.*, 2011, pp. 1–10.
8. J. Ludewig, "Models in software engineering: An introduction," *SoSyM*, vol. 2, no. 1, pp. 5–14, 2003, doi: 10.1007/s10270-003-0020-3.
9. M. Brambilla, J. Cabot, and M. Wimmer, "Model-driven software engineering in practice," in *Synthesis Lectures on Software Engineering*, 1st ed., L. Baresi, Ed. San Rafael, CA, USA: Morgan & Claypool, 2012.
10. P.-A. Muller, F. Fondement, B. Baudry, and B. Combemale, "Modeling modeling modeling," *Softw. Syst. Model.*, vol. 11, no. 3, pp. 347–359, 2012, doi: 10.1007/s10270-010-0172-x.
11. "Educators symposium," in *Proc. ACM/IEEE Int. Conf. Model Driven Eng. Lang. Syst. Companion (MODELS-C)*, Fukuoka, Japan, Oct. 10–15, 2021. [Online]. Available: https://conf.researchr.org/track/models-2021/models-2021-educators-symposium
12. T. Gorschek, E. Tempero, and L. Angelis, "On the use of software design models in software development practice: An empirical investigation," *J. Syst. Softw.*, vol. 95, p. 193, Sep. 2014, doi: 10.1016/j.jss.2014.03.082.
13. R. Heldal, P. Pelliccione, U. Eliasson, J. Lantz, J. Derehag, and J. Whittle, "Descriptive vs prescriptive models in industry," in *Proc. ACM/IEEE 19th Int. Conf. Model Driven Eng. Lang. Syst., (MoDELS)*, 2016, pp. 216–226, doi: 10.1145/2976767.2976808.
14. J. Hutchinson, J. Whittle, and M. Rouncefield, "Model-driven engineering practices in industry: Social, organizational and managerial factors that lead to success or failure," *Sci. Comput. Program.*, vol. 89, pp. 144–161, Sep. 2014, doi: 10.1016/j.scico.2013.03.017.
15. G. Cockton, M. Lárusdóttir, P. Gregory, and Å. Cajander, "Integrating user-centered design in agile development," in *Human-Computer Interaction Series*. Cham: Springer-Verlag, 2016, pp. 1–46.
16. D. Teichroew and H. Sayani, "Automation of system building," *Datamation*, vol. 17, no. 16, pp. 25–30, 1971.
17. D. W. Embley, S. Liddle, and O. Pastor, "Conceptual-model programming: A manifesto," in *Handbook of Conceptual Modeling*, D. Embley and B. Thalheim, Eds. Berlin: Springer-Verlag, 2011, pp. 3–16.
18. E. W. Dijkstra, "Chapter EWD227: Stepwise program construction," in *Selected Writings on Computing: A Personal Perspective*. New York, NY, USA: Springer-Verlag, 1982, pp. 1–14.

OSCAR PASTOR is the director of internationalization and transference at the Valencian Research Institute for Artificial Intelligence, Universitat Politecnica de Valencia, Valencia, 46009, Spain. Contact him at opastor@dsic.upv.es.

ALFONSO PIERANTONIO is a full professor at the University of L'Aquila, L'Aquila, 67100, Italy. Contact him at alfonso.pierantonio@univaq.it.

GUSTAVO ROSSI is a professor at La Plata National University, La Plata, Argentina, and a researcher at the National Scientific and Technical Research Council, Buenos Aires, 1900, Argentina. Contact him at gustavo@lifia.info.unlp.edu.ar.