What is the Future of Modelling?

Antonio Bucchiarone, Federico Ciccozzi, Leen Lambers, Alfonso Pierantonio, Matthias Tichy, Massimo Tisi, Andreas Wortmann, Vadim Zaytsev

What is the Future of Modelling?

Antonio Bucchiarone, Federico Ciccozzi, Leen Lambers, Alfonso Pierantonio, Matthias Tichy, Massimo Tisi, Andreas Wortmann, and Vadim Zaytsev

***Abstract*—Modelling languages and frameworks have been the key technology for advancing Model-Driven Engineering (MDE) methods and tools. Many industrial and research tools have been realised and are used across many domains. Hence, we think it is the right time to define what should be the future of modelling technologies, especially the requirements for the next generation of modelling frameworks and languages.**

In January 2020, the Second Winter Modelling Meeting (WMM2020) was held in San Vigilio di Marebbe (Italy), focusing on the analysis of the state of research, state of practice, and state of the art in MDE. The event brought together experts from industry, academia, and the open-source community to assess (i) what had changed in research on modelling in the last ten years, (ii) which problems are still unsolved, and (iii) which new challenges have arisen.

This article presents a set of success stories and driving success factors of modelling and MDE, as well as a set of challenges and corresponding research directions that emerged from the synthesis of the results of our analysis.

***Index Terms*—Software Modelling, Success Stories, Research Roadmap**

I. Introduction

The use of models in computer science can be traced back to the earliest efforts in the field. The sequences of designs by Charles Babbage on his "Analytical Machine"[1] were the first models of a Turing-complete mechanical device. Since then, many different modelling languages have been designed in software engineering and strongly shaped the discipline of Model-Driven Engineering (MDE)[2].

The role of models in improving productivity in software engineering is a recurring theme. During the "Peak of Inflated Expectations" in the hype cycle[3] at the beginning of the 2000s, concepts like Model Driven Architecture (MDA), Model-Driven Software Engineering (MDSE), as well as the promotion of the Unified Modelling Language (UML) as the panacea for all possible problems in software engineering, have substantially influenced the MDE discipline.

Since then, software modelling has arrived at the "Plateau of Productivity," and the modelling community learned when and how to use its founding principles for improving the productivity of software engineering [8].

The aim of this report is (1) to present three success stories where modelling has been applied in various ways for different target groups to achieve other goals, and (2) to formulate a set of areas and corresponding research directions.

II. Modelling Success Stories

In the following sections, we review successful shapes of modelling: *model-based systems engineering*, *low-code software development*, and *informal software modelling*. Each of the three sections concludes with a summary of the success factors relevant for that modelling shapes illustrated in Figure 1.

A. Model-Based Systems Engineering

For the systematic and reliable engineering of (cyber-physical) systems, modelling, as a technique to describe or prescribe the system's properties under development, is the essential foundation. Consequently, engineers from various domains have been modelling for ages. For instance, electrical engineers use mathematical formulae to describe a system's processes, and mechanical engineers use technical drawings to prescribe constructions.

Antonio Bucchiarone is with the Fondazione Bruno Kessler, Italy
Federico Ciccozzi is with Mälardalen University, Sweden
Leen Lambers is with the HPI, University of Potsdam, Germany
Alfonso Pierantonio is with Università degli Studi dell'Aquila, Italy
Matthias Tichy is with Ulm University, Germany
Massimo Tisi is with the Institut Mines-Telecom Atlantique, France
Andreas Wortmann is with the RWTH Aachen, Germany
Vadim Zaytsev is with the University of Twente, The Netherlands

[1] https://en.wikipedia.org/wiki/Analytical_Engine

[2] See also https://modeling-languages.com

[3] https://en.wikipedia.org/wiki/Hype_cycle

Fig. 1. Modelling Success Stories and Challenges.

The automated analysis and synthesis of system models and their parts have become possible thanks to modeling notations that helped define and establish practices in application areas (e.g., automotive, railway, aerospace).

Via model-based analyses, simulations, and syntheses across system parts provided by experts of different domains, challenges and mistakes too costly to address in real systems can be uncovered early. For example cases like extreme situations (cf. Boeing 737 MAX) or incompatible assumptions of system parts (cf. Ariane 5), if proactively discovered, can significantly reduce failures in the resulting systems.

Research and industry have produced various modelling techniques for the engineering of software for embedded and cyber-physical systems. Those techniques address different phases in the engineering process. Large systems engineering companies in avionics successfully apply architecture modelling languages to decompose system components' structure and behaviour and facilitate development, analysis, and integration of subcomponents across multiple departments. Particularly, the modelling and analysis of extra-functional requirements like dependability and timeliness have been a success in that area.

To describe continuously varying behaviour, corresponding modelling languages have been broadly and successfully adopted. In many engineering departments, MatLab's Simulink[4] has become one of the prime modelling tools. Modelling languages enable both the automated analysis of correctness and other properties and automatic generation of code that is widely used in today's products like Eclipse, MetaEdit+, and Modelica.

The standardisation of modelling languages by ANSI/ISA, NIST, or ISO has built the foundation for multi-stakeholder, cross-company modelling required to successfully engineer cyber-physical systems. Popular standards that employ modelling techniques, define modelling languages, such as function block diagrams (IEC 61131-3[5]), IDEF0 manufacturing functions (NIST FIPS 183[6]), or the EXPRESS data modelling language (ISO 10303[7]). These standards allow companies and stakeholders to rely on the same explicit models to ensure the systems' compatibility under development. Moreover, they enable tool builders to rely on stable shared foundations. Overall, standards are vital to the success of modelling in industrial practice.

The success of modelling for cyber-physical systems is due to the levels of precision of the modelling languages, standardisation, and the im-

[4]https://www.mathworks.com/products/simulink.html
[5]https://plcopen.org/status-iec-61131-3-standard
[6]https://csrc.nist.gov/publications/fips
[7]https://www.iso.org/standard/38047.html

portance of frontloading in systems engineering: successful languages, such as AADL[8] or Simulink, are tailored to broad domains without being overly specific.

Moreover, this broad use of such sufficiently precise modelling languages fosters communication, understanding, and development across experts from different departments, companies, and domains. Another reason for the success of explicit modelling in engineering is that modelling languages—and, by extension, the tools featuring them—are either supported by large industrial consortia or standardised. The broad commitment to specific technologies, languages, and standards enables companies to rely on their availability and stability in the future, which encourages further commitments to their use, development, and extension. However, this generality introduces a challenging conceptual gap [7] between the experts' domains, with their concepts and methods, and the solution domain of software engineering, along with its own concepts and methods; this gap needs to be addressed in the next generation modelling tools. The idea of bridging these two worlds with automated means is intriguing and has engaged the community, but it has not always worked. Successful applications in these areas focus on the domains' specifics and provide well-integrated platforms with clear technical benefits for the developers.

Heavyweight modelling, profoundly relying on rigorous specification and state of the art engineering practices, as employed in cyber-physical systems, works well if models are continually used throughout the development process. Ideally, models provide the single source of truth; tools are used to analyse the models, performing timing analysis, correctness, control stability of feedback controllers, easy deployment, code generation, or interpretation of the model.

In **Model-Based Systems Engineering**, modelling is successful if standardised or established modeling techniques and languages lead to cost- and time-savings for domain experts and system integrators. This is often achieved by enabling analysis and synthesis of system parts at design time, long before the real system or its components are manufactured, in order to avoid failures, rather than to detect reasons for them.

These benefits are also shown in the highly positive results of empirical studies on the effect of modelling in the embedded systems domain, where modelling affects very positively the productivity of engineers and the quality of the products [8].

B. Low-Code Software Development

Recently, low-code development platforms have been considered promising for democratising digital processes in organisations [4]. Notably, earlier attempts to simplify software development, e.g., fourth-generation languages (4GLs), were not exempt from problems. Nowadays, end-user programming, mashups, or situational programming leverage abstraction and automation by making full use of recent advances in domain-specific modeling, visual editing, and user experience. Spreadsheet applications' overwhelming success inspired low-code platforms with their ease of use and substantial computational power. The major market analysis firms have highlighted the current impressive investments by vendors and customers in low-code platforms for business applications, and foreseen a positive trend for the next years. Besides the current commercial success in business application development, other domains are considered reasonable grounds for these solutions, like knowledge management and digital transformation in manufacturing.

Several web giants have recently started providing their own low-code development platform (Microsoft PowerApps[9], Google App Maker[10], Amazon Honeycode[11]). Some medium-sized vendors were recently protagonists of impressive acquisitions (e.g., $360 million by KKR and Goldman Sachs for OutSystems[12], and $730 million by Siemens for Mendix[13]). Other popular low-code platform providers include AppSheet[14], Caspio[15], FileMaker[16], Kony[17], Parabola[18], QuickBase[19] and

[8]http://www.aadl.info

[9]https://powerapps.microsoft.com
[10]https://developers.google.com/appmaker
[11]https://www.honeycode.aws
[12]https://www.outsystems.com
[13]https://www.mendix.com
[14]https://www.appsheet.com
[15]https://www.caspio.com
[16]https://www.filemaker.com
[17]https://www.kony.com
[18]https://parabola.io
[19]https://www.quickbase.com

Salesforce[20].

The prominent success of low-code platforms in business applications is tied to the present-day software production landscape. Despite aggressive recruitment efforts and innovative working conditions, the IT industry's development capability is at capacity. Low-code development platforms enable the inclusion of non-professional developers into the application production process, letting IT experts to focus on the more knowledge-intensive tasks.

Typically, low-code platforms are inspired by different modeling *paradigms* and tailored to the most diverse domains. Therefore, it is not trivial to provide a unifying and informative characterisation of the features they offer and for which types of applications. Nevertheless, they can be distinguished by the following factors: *(1)* advanced user-interfaces that help the user develop his/her proficiency with the tool in a learning-by-doing manner quickly; *(2)* Platform-as-a-Service architecture to mitigate the accidental complexity of managing (e.g., installing and upgrading versioned components) the modelling environment, deploying the application, and monitor its execution. Finally, *(3)* machine-learning techniques to ease the user's development process by providing him with automatic assistance tools, such as a recommendation system. Modelling languages and model-driven techniques are used within low-code platforms either explicitly or implicitly (e.g., hidden behind visual editors or forms).

Low-code development platforms' technical merits do not fully explain their significant commercial success thoroughly. While their interoperability, openness, and scalability are still subject to further investigations [13], their accessibility, user focus, strategies for hiding accidental complexity, and a convenient learning curve can spur innovation leading to better model-driven platforms.

In **Low-Code Software Development**, modelling is successful if it enables software engineering with a minimal upfront investment in setup (e.g., by native integration with an existing development platform), training (e.g., by advanced user interfaces, self-explaining and AI-assisted IDEs), and deployment (e.g., as Platform-as-a-Service), and with costs rising in proportion to the business value of the developed applications.

C. Informal Modelling

The above two scenarios show the application and value of modelling in two specific and different application areas. Additionally, modelling is also extensively used in generic software engineering albeit with different intentions than the applications above.

Störrle conducted a survey [12] to identify how and with which frequency modelling is used in the software engineering industry. The results show that over 70% of all survey participants used models often or always for communicative and cognitive processes, which were the most popular usage areas; code generation was never or rarely used by half of the survey's population. Additionally, models were used more in the early phases of the development, e.g., for domain- and requirements-oriented discussions. This means that models were used more as a "thought tool" and to facilitate discussions among stakeholders with diverse backgrounds. According to the survey results, software architects were the stakeholders benefiting the most from modelling, with 91% of the respondents rating the benefits for software architects as "a lot" or "crucial".

Another empirical study by Baltes and Diehl [1] showed that informal modelling such as sketching is frequent in software engineering. 77% of the study participants created and/or used model sketches in the previous week, and 68% of the sketches were rated as informal. In line with the findings of Störrle, Baltes and Diehl reported that design, explaining or understanding were the most common purposes of the sketches. Similarly, sketches were often used to analyse requirements. Sketches are not only used as informal and temporary means for communication and discussion; whiteboard sketches are sometimes subsequently detailed on paper, later more formally modelled in a tool and sometimes converted to text too. Interestingly, also generic drawing tools like yEd[21], miro[22] or Visio are reported to be used for modelling. Finally, the study reported that about half the sketches were rated as helpful to understand the related source code artefact(s) in the future.

[20]https://www.force.com

[21]https://www.yworks.com/products/yed

[22]https://miro.com

An example of informal modelling via collaborative sketching on an interactive whiteboard is brought by OctoUML[23], which supports the creation of UML models at various levels of formality (or precision), collaborative and multi-modal interaction. OctoUML is a prototype of a new generation design environment that enhances informal collaboration when architecting a product. Similar approaches are beneficial in other settings, such as teaching and training.

In contrast to formal and, more generally, less flexible modelling, *in*formal modelling (e.g., by sketching on a whiteboard) is particularly useful for communication, collaboration, and understanding. Here, cheap solutions like whiteboard or drawing tools are enough to reap significant benefits and make the design phase faster and more effective without high investments. Moreover, being more flexible and less chained to specific formalisms and constraints, informal modelling pushes down the learning/training curve of beginners and supports the needs of a wider variety of stakeholders. Industrial experience using modelling tools shows a noteworthy division between stakeholders preferring different types of modelling notations; we believe that informal and thereby flexible modelling is the only viable way to broaden acceptance of modelling tools by industrial stakeholders.

..

In **Informal Modelling**, modelling is successful if a wide variety of stakeholders can employ it for communicative and cognitive processes in early development phases using emergent and flexible graphical notations while postponing any structural limitations on the sketched models as long as possible.

III. Challenges and Opportunities

There have been many empirical studies carried out in modelling-rich domains about standing issues, apparent trends, and future challenges: object-oriented modelling, business process modelling, model-data management, self-adaptive systems, and specifically in MDSE [2]. Many mention the same issues, but in different contexts: demonstration of added value to potential users/customers, integration of produced artefacts, learnability, reliability, etc. In the following, we highlight the most important new issues that have changed significantly over the last few years, as discussed during WMM2020.

A. Artificial Intelligence

Admittedly, it was impossible to overlook the recent advances in Artificial Intelligence (AI), which are now dramatically changing how we design, engineer, and maintain software. Many believe that this will cause a massive shift in a skill set that software developers are expected to carry. MDE techniques are also being enhanced with AI extensions for automation and bringing quantifiable advantages [5]. The main reason is that many MDE techniques are already based on the intensive use of knowledge and data. Even success factors of other areas of AI like TensorFlow[24] and its internal DSL, are similar to those of software modelling. In the future, we will only see more applications like modelling bots that assist modellers by identifying potential issues and giving advice [5] or model recommenders integrated into IDEs [6]. For example, the new version of Matlab Simulink includes reusing components by creating library blocks from subsystem clones and replacing clones with library links.

B. Multi-Paradigm Modelling

When engineering cyber-physical systems, experts from different domains collaborate to contribute solutions to various aspects of the systems under development. To engineer these solutions, experts employ different paradigms (discrete vs. continuous, geometric vs. functional, ...), reified in modeling languages, tool, and processes, that need to be integrated to describe the systems under development. One of the solutions to this integration is known as Multi-Paradigm Modelling (MPM) [10], which envisions to "model everything", i.e., each aspect of the system and each corresponding process is specified using models at the appropriate level of abstraction, while model transformations propagate information. By modeling everything, the paradigms (models plus related processes) provided by different domain experts are made explicit and can be integrated, analyzed, and synthesized automatically. This ultimately enables to support cross-disciplinary communication and collaboration.

[23]https://github.com/Imarcus/OctoUML

[24]https://www.tensorflow.org

TABLE I
SELECTED KEY RESULTS OF MODELLING SUCCESS STORIES

Success Story	Modeling Goal	Experiences	Ref.
Model-Based Systems Engineering	Cost- and time-savings for domain experts and system integrators	"[MBSE] enables realization of several key benefits including: Establishing a common understanding of the structure and meaning of information; Enabling domain knowledge reuse; Making domain assumptions explicit; Maintaining separation of domain knowledge and operational knowledge; Supporting reasoning and analysis of domain knowledge; Capturing agreements on usage; Enabling consistent [...] conversation, thereby preventing confusion and misunderstanding."	[9]
Low-Code Software Development	Shifting programming tasks from software engineers to domain experts	"[...] a significant difference from traditional development was observed in that the application architecture was provided through the platform, leaving the developer to concentrate primarily on what data is required and how it should be captured."	[14]
Informal Modeling	Communication & collaboration	"over 70% of all survey participants used models often or always for communicative and cognitive processes" "models were used more as a "thought tool" and to facilitate discussions among stakeholders with diverse backgrounds"	[12]

C. Adoption Model for Modelling

As outlined in Section II, different types of modelling can be successful for various reasons, in different domains, and with different characteristics. Another challenge that we therefore identified is how to support organisations on their ways towards applying modelling successfully. In other words, how can an organisation assess, evaluate, and improve its modelling activities?

An initial idea for coming up with a concrete solution to this overall challenge is elaborating an *adoption model for modelling*. This idea is closely related to the development of a maturity model for model-driven development initiated by Rios et al. [11].

The overall goal of the adoption model will be to provide guidelines for discovering the right level of adoption of modelling within (different parts of) an organisation as well as to support the transitioning from one level to the next when needed.

Implementing such guidelines goes along with appropriate training and education. Teaching modelling is commonplace and almost any degree program offers courses at different levels that range from foundational aspects to laboratory practice. Recently, the state-of-the-practice of modelling and MDE has been characterized in [3], where a precise picture of the covered topics and their relevance is presented. In particular, it emerges that modelling can be considered a *trait d'union* between software and language engineering: on the one hand, models can be used at any stage of the development process for documenting, analyzing, designing, deploying and simulating systems; on the other hand, metamodelling techniques can be used for designing notations and the associated modelling environment. Consequently, this distinction makes place for a diversity of courses at both bachelor and master level covering the most relevant aspects of model-based software engineering and model-driven engineering.

D. Model Management

Current modelling tools are sophisticated tools that provide features to simplify and automate development activities. However, the sheer complexity of modern software systems often requires designers to deal with heterogeneous collections of related models. Their continuous management, deployment, and integration are crucial at different development stages and can take the form of reusing artifacts, analysing their characteristics, managing consistency or leveraging the underlying informative contents. Several model repositories have been proposed over the last decade. A daunting challenge is represented by the enhancement of these platforms from merely cloud storage to (possibly collaborative) modelling environments where the designer can smoothly maintain collections of artifacts consistent, efficiently cluster, quickly locate and reuse them, and compose different transformations. More recently, several initiatives, including Visual Studio

Code[25], Eclipse Che[26], Theia[27] and others, hold great promise to shift modelling environments from monolithic installations to cloud-based platforms to reduce the accidental complexity and extend the set of offered features. In a way, this represents a refreshing scenario where commercial competitors will also contribute to the field's advance. As a consequence, it seems that state-of-the-art suggests that versioning tools will soon be part of the modeling environment alongside collaborative modeling possibilities.

IV. Conclusion

Figure 1 provides a visual summary of this paper. It captures what we discussed at a single glance: *how modelling is being increasingly adopted across the diverse areas of software and system engineering, and beyond*. Besides fields where models are traditional instruments, like embedded and cyber-physical systems domains, new areas of applications emerged, including the so-called low-code development platforms where even people with considerably less programming experience can develop software applications within organizations. Empirical studies showed that modelling positively affects both engineers' productivity and the products' resulting quality, also thanks to consortia and standardisation bodies.

The urge for improved platforms and foundations is stringent because of the pervasive adoption of models and related environments. The paper presented some of the most daunting challenges to move towards a community roadmap. Such a roadmap aims to provide a motivated collection of challenges, addressing which can be used to improve modelling technology and leverage adjacent research and development fields.

Acknowledgment

The authors that compiled this document would like to thank all participants of the Second Winter Modelling Meeting[28], especially those that actively contributed to discussing these issues and describing them.

[25]https://code.visualstudio.com
[26]https://www.eclipse.org/che/
[27]https://theia-ide.org
[28]http://eventmall.info/WMM2020/

REFERENCES

[1] S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014*, S. Cheung, A. Orso, and M. D. Storey, Eds. ACM, 2014, pp. 530–541. [Online]. Available: https://doi.org/10.1145/2635868.2635891

[2] A. Bucchiarone, J. Cabot, R. F. Paige, and A. Pierantonio, "Grand challenges in model-driven engineering: an analysis of the state of the research," *Software and Systems Modeling*, vol. 19, no. 1, pp. 5–13, 2020. [Online]. Available: https://doi.org/10.1007/s10270-019-00773-6

[3] L. Burgueño, F. Ciccozzi, M. Famelis, G. Kappel, L. Lambers, S. Mosser, R. F. Paige, A. Pierantonio, A. Rensink, R. Salay *et al.*, "Contents for a model-based software engineering body of knowledge," *Software and systems modeling*, vol. 18, no. 6, pp. 3193–3205, 2019.

[4] R. D. Caballar. Programming Without Code: The Rise of No-Code Software Development. IEEE Spectrum. [Online]. Available: https://spectrum.ieee.org/tech-talk/computing/software/programming-without-code-no-code-software-development

[5] J. Cabot, R. Clarisó, M. Brambilla, and S. Gérard, "Cognifying model-driven software engineering," in *Software Technologies: Applications and Foundations*, M. Seidl and S. Zschaler, Eds. Cham: Springer International Publishing, 2018, pp. 154–160.

[6] A. Dyck, A. Ganser, and H. Lichter, "A framework for model recommenders requirements, architecture and tool support," in *2014 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014, pp. 282–290.

[7] R. France and B. Rumpe, "Model-driven Development of Complex Software: A Research Roadmap," *Future of Software Engineering (FOSE '07)*, pp. 37–54, May 2007.

[8] G. Liebel, N. Marko, M. Tichy, A. Leitner, and J. Hansson, "Model-based engineering in the embedded systems domain: an industrial survey on the state-of-practice," *Software and Systems Modeling*, vol. 17, no. 1, pp. 91–113, 2018. [Online]. Available: https://doi.org/10.1007/s10270-016-0523-3

[9] A. M. Madni and M. Sievers, "Model-based systems engineering: Motivation, current status, and research opportunities," *Systems Engineering*, vol. 21, no. 3, pp. 172–190, 2018.

[10] P. J. Mosterman and H. Vangheluwe, "Computer automated multi-paradigm modeling: An introduction," *SIMULATION*, vol. 80, no. 9, pp. 433–450, 2004.

[11] E. Rios, T. Bozheva, A. Bediaga, and N. Guilloreau, "MDD maturity model: A roadmap for introducing model-driven development," in *Proceedings of the Second European Conference on Model Driven Architecture — Foundations and Applications (ECMDA-FA)*, ser. Lecture Notes in Computer Science, A. Rensink and J. Warmer, Eds., vol. 4066. Springer, 2006, pp. 78–89.

[12] H. Störrle, "How are conceptual models used in industrial software development?: A descriptive survey," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE 2017, Karlskrona, Sweden, June 15-16, 2017*, E. Mendes, S. Counsell, and K. Petersen, Eds. ACM, 2017, pp. 160–169. [Online]. Available: https://doi.org/10.1145/3084226.3084256

[13] M. Tisi, J.-M. Mottu, D. S. Kolovos, J. De Lara, E. M. Guerra, D. Di Ruscio, A. Pierantonio, and M. Wimmer, "Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms," in *STAF 2019 Co-Located Events Joint Proceedings*, ser. CEUR Workshop Proceedings (CEUR-WS.org), Eindhoven, Netherlands, Jul. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02363416

[14] R. L. Totterdale, "Case study: The utilization of low-code development technology to support research data collection." *Issues in Information Systems*, vol. 19, no. 2, 2018.