[BCL+21] A. Bucchiarone, F. Ciccozzi, L. Lambers, A. Pierantonio, M. Tichy, M. Tisi, A. Wortmann, V. Zaytsev:
{What Is the Future of Modeling?
In: IEEE software, 38(2), pp. 119-127, IEEE, March-April 2021.
www.se-rwth.de/publications/

The goal of this article is twofold: 1) to present three success stories where modeling 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.

## Modeling Success Stories

In the following sections, we review the successful shapes of modeling: model-based systems engineering (MBSE), low-code software development, and informal software modeling. Each of the three sections concludes with a summary of the success factors relevant for the modeling shapes shown in Figure 1 with key experiences for each of the shapes listed in Table 1.

### MBSE

For the systematic and reliable engineering of (cyberphysical) systems, modeling, as a technique used to describe or prescribe the system's properties under development, is the essential foundation. Consequently, engineers from various domains have been modeling for ages. For instance, electrical engineers use mathematical formulae to describe a system's processes, and mechanical engineers use technical drawings to prescribe constructions (see Table 1).

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, and 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 (*Boeing 737 MAX*) or incompatible assumptions of system parts (*Ariane 5*), if proactively discovered, can significantly reduce failures in the resulting systems.

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

To describe continuously varying behavior, corresponding modeling languages have been broadly and successfully adopted. In many engineering departments, MATLAB's Simulink (https://www.mathworks.com/products/simulink.html) has become one of the prime modeling tools. Modeling languages enable

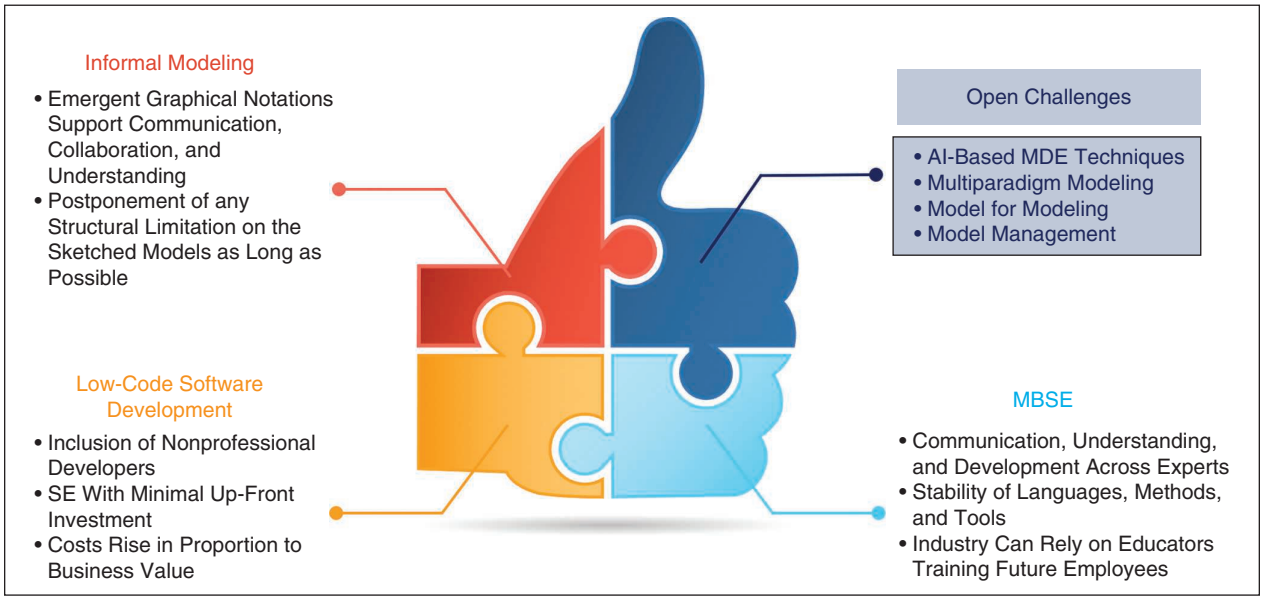


**FIGURE 1.** Modeling success stories and challenges. AI: artificial intelligence.

both the automated analysis of correctness and other properties and the automatic generation of code that is widely used in today's products such as Eclipse, MetaEdit+, and Modelica.

The standardization of modeling languages by ANSI/International Society of Automation, the National Institute of Standards and Technology (NIST), or the International Organization for Standardization has built the foundation for multistakeholder, cross-company modeling required to successfully engineer cyberphysical systems. Popular standards that employ modeling techniques define modeling languages, such as function block diagrams [IEC 61131-3 (https://plcopen.org/status-iec-61131-3-standard)], IDEF0 manufacturing functions [NIST FIPS 183 (https://csrc.nist.gov/publications/fips)], or the EXPRESS data modeling language [ISO 10303 (https://www.iso.org/standard/38047.html). 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 modeling in industrial practice.

The success of modeling for cyberphysical systems is due to the levels of precision of the modeling languages, standardization, and the importance of frontloading in systems engineering: Successful languages, such as AADL (http://www.aadl.info) or Simulink, are tailored to broad domains without being overly specific.

Moreover, this broad use of such sufficiently precise modeling languages fosters communication, understanding, and development across experts from different departments, companies, and domains. Another reason for the success of explicit modeling in engineering is that modeling languages—and, by extension, the tools featuring them—are either supported by large industrial consortia or are standardized. 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[3] between the experts' domains, with their concepts and methods, and the solution domain of SE, along with its own concepts and methods; this gap needs to be addressed in next-generation modeling 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 developers.

**Table 1. The selected key experiences of modeling success stories.**

| Success story | Modeling goal | Experiences | Reference |
|---|---|---|---|
| MBSE | 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, and enabling consistent [...] conversation, thereby preventing confusion and misunderstanding." | 14 |
| 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 are required and how they should be captured." | 15 |
| Informal modeling | Communication and 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." | 6 |

## MODEL-BASED SOFTWARE ENGINEERING

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

Heavyweight modeling, 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 analyze the models, performing timing analysis, correctness, control stability of feedback controllers, easy deployment, code generation, or interpretation of the model (see "Model-Based Software Engineering").

These benefits are also shown in the highly positive results of empirical studies on the effect of modeling in the embedded systems domain, where modeling affects very positively the productivity of engineers and the quality of the products.[2]

### Low-Code Software Development

Recently, low-code development platforms have been considered promising for democratizing digital processes in organizations.[4] Notably, earlier attempts to simplify software development, e.g., fourth-generation languages, were not exempt from problems. Today, 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 has inspired low-code platforms, with their ease of use and substantial computational power. Major-market analysis firms have highlighted current impressive investments by vendors and customers in low-code platforms for business applications, an foreseen positive trend for the next several years. Besides the current commercial success in business application development, other domains are considered reasonable grounds for these solutions, such as knowledge management and digital transformation in manufacturing.

Several web giants have recently started providing their own low-code development platform, e.g., Microsoft PowerApps (https://powerapps.microsoft.com), Google App Maker (https://developers.google.com/appmaker), and Amazon Honeycode (https://www.honeycode.aws). Some medium-sized vendors were recently protagonists of impressive acquisitions [e.g., US$360 million by KKR and Goldman Sachs for OutSystems (https://www.outsystems.com) and US$730 million by Siemens for Mendix (https://www.mendix.com)]. Other popular low-code platform providers include AppSheet (https://www.appsheet.com), Caspio (https://www.caspio.com), FileMaker (https://www.filemaker.com), Kony (https://www.kony.com), Parabola (https://parabola.io), QuickBase (https://www.quickbase.com), and Salesforce (https://www.force.com).

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 nonprofessional developers into the application production process, letting IT experts 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 characterization 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 or 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 modeling environment, deploying the application, and monitoring its execution; and 3) machine learning techniques to ease the user's development process by providing him or her with automatic assistance tools, such as a recommendation system. Modeling 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. Although their interoperability, openness, and scalability are still subject to further investigation,[5] their accessibility, user focus, strategies for hiding accidental complexity, and a convenient learning curve can spur innovation, leading to better model-driven platforms (see "Low-Code Software Development").

### Informal Modeling

The aforementioned two scenarios show the application and value of modeling in two specific and different application areas. Additionally, modeling is also used extensively in generic SE, albeit with different intentions than the applications mentioned previously.

Störrle conducted a survey[6] to identify how and with which frequency modeling is used in the SE industry. The results show that more than 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 who benefited the most from modeling, with 91% of the respondents rating the benefits for software architects as "a lot" or "crucial."

Another empirical study by Baltes and Diehl[7] showed that informal modeling such as sketching is frequent in SE. Of the study participants, 77% 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 analyze 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 modeled in a tool and sometimes converted to text, too. Interestingly, generic drawing tools like yEd (https://www.yworks.com/products/yed), miro (https://miro.com), or Visio were reported to have been used for modeling. Finally, the study reported that roughly half of the sketches were rated as "helpful" to understand the related source code artefact(s) in the future.

An example of informal modeling via collaborative sketching on an interactive whiteboard is introduced by OctoUML (https://github.com/Imarcus/OctoUML), which supports the creation of UML models at various levels of formality (or precision), collaborative, and multimodal interaction. OctoUML is a prototype of a new-generation design environment

## LOW-CODE SOFTWARE DEVELOPMENT

In low-code software development, the modeling is successful if it enables software engineering with a minimal up-front investment in setup (e.g., by native integration with an existing development platform), training (e.g., by advanced user interfaces, self-explaining, and artificial intelligence-assisted integrated development environments), and deployment (e.g., as platform-as-a-service), and, if costs rise in proportion to the business value of the developed applications.

> Modeling languages and model-driven techniques are used within low-code platforms either explicitly or implicitly (e.g., hidden behind visual editors or forms).

## INFORMAL MODELING

Informal modeling 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.

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 modeling, informal modeling (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 costly investments. Moreover, being more flexible and less chained to specific formalisms and constraints, informal modeling pushes down the learning/training curve of beginners and supports the needs of a wider variety of stakeholders. Industrial experience using modeling tools shows a noteworthy division between stakeholders preferring different types of modeling notations; we believe that informal and thereby flexible modeling is the only viable way to broaden the acceptance of modeling tools by industrial stakeholders (see "Informal Modeling").

## Challenges and Opportunities

There have been many empirical studies carried out in modeling-rich domains about standing issues, apparent trends, and future challenges, including object-oriented modeling, business process modeling, model-data management, self-adaptive systems, and specifically in MDSE.[8] Many mention the same issues but in different contexts: the demonstration of added value to potential users/customers, and the integration of produced artifacts, learnability, reliability, and so forth. In the following, we highlight the most important new issues that have changed significantly over the last few years, as discussed during the Second Winter Modeling Meeting (WMM2020).

### AI

Admittedly, it was impossible to overlook the recent advances in AI, which are now dramatically changing how we design, engineer, and maintain software. Many believe that this will cause a massive shift in skill sets, which software developers will be expected to carry. MDE techniques are also being enhanced with AI extensions for automation and bringing quantifiable advantages.[9] The main reason is that many MDE techniques are already based on the intensive use of knowledge and data. Even the success factors of other areas of AI, like TensorFlow (https://www.tensorflow.org) and its internal domain specific language, are similar to those of software modeling. In the future, we will see more applications, such as modeling bots that assist modelers by identifying potential issues and giving advice,[8] or model recommenders that are integrated into IDEs.[10] For example, the new version of MATLAB Simulink includes reusing components by creating library blocks from subsystem clones and replacing clones with library links.

### Multiparadigm Modeling

When engineering cyberphysical 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 versus continuous, geometric versus functional, and so on) reified in modeling languages; tools; and processes; which need to be integrated to describe the systems under development. One of the solutions to this integration is known as *multiparadigm modeling*,[11] which envisions modeling 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 the support of cross-disciplinary communication and collaboration.

### Adoption Model for Modeling

As outlined in the "Modeling Success Stories" section, different types of modeling can be successful for various reasons, in different domains, and with different characteristics. Another challenge that we have therefore identified is how best to support organizations on their

way toward applying modeling successfully. In other words, how can an organization assess, evaluate, and improve its modeling activities?

An initial idea for coming up with a concrete solution to this overall challenge is elaborating an adoption model for modeling. This idea is closely related to the development of a maturity model for the model-driven development initiated by Rios et al.[12]

The overall goal of the adoption model will be to provide guidelines for discovering the right level of adoption of modeling within (different parts of) an organization 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 modeling is commonplace and almost any degree program offers courses at different levels, which range from

## ABOUT THE AUTHORS


**ANTONIO BUCCHIARONE** is a senior researcher at the Motivational Digital Systems Unit, the Fondazione Bruno Kessler, Trento, 38123, Italy. Further information about him can be found at https://bucchiarone.bitbucket.io/. Contact him at bucchiarone@fbk.eu.

**MATTHIAS TICHY** is a full professor for software engineering for complex, technical systems at Ulm University, Ulm, 89081, Germany, and the director of the Institute of Software Engineering and Programming Languages, Ulm, 89081, Germany. Further information about him can be found at https://www.tichy.de. Contact him at matthias.tichy@uni-ulm.de.

**FEDERICO CICCOZZI** is an associate professor at Mälardalen University, Västerås, 722 20, Sweden. Further information about him can be found at http://www.es.mdh.se/staff/266-Federica_Ciccozzi. Contact him at federico.ciccozzi@mdh.se.

**MASSIMO TISI** is an associate professor at the Department of Computer Science, Institut Mines-Telecom Atlantique, Nantes, 44300, France, and the deputy leader of the NaoMod team at the Laboratory of Digital Sciences of Nantes, a joint research unit of the French National Centre for Scientific Research, Nantes, 44300, France. Further information about him can be found at https://massimotisi.github.io/. Contact him at massimo.tisi@imt-atlantique.fr.

**LEEN LAMBERS** is a senior researcher at the System Analysis and Modeling Group, Hasso Plattner Institute, University of Potsdam, Potsdam, 14482, Germany. Further information about her can be found at https://hpi.de/en/giese/staff/dr-leen-lambers.html. Contact her at leen.lambers@hpi.de.

**ANDREAS WORTMANN** is a senior researcher and the chair for software engineering at RWTH Aachen University, Aachen, 52062, Germany. Further information about him can be found at http://www.wortmann.ac. Contact him at wortmann@se-rwth.de.

**ALFONSO PIERANTONIO** is a professor in computer science at the Università degli Studi dell'Aquila, L'Aquila, 67100, Italy. Further information about him can be found at https://pieranton.io/. Contact him at alfonso.pierantonio@univaq.it.

**VADIM ZAYTSEV** is an associate professor of software evolution at the University of Twente, Enschede, 7522 NB, The Netherlands. Further information about him can be found at http://grammarware.net. Contact him at vadim@grammarware.net.

foundational aspects to laboratory practice. Recently, the state of the practice of modeling and MDE has been characterized by Burgueño et al.,[13] where a precise picture of the covered topics and their relevance is presented. In particular, it emerges that modeling 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; while on the other hand, metamodeling techniques can be used for designing notations and the associated modeling environment. Consequently, this distinction makes for a diversity of courses at both the bachelor and master's level, covering the most relevant aspects of model-based SE and MDE.

### Model Management

Current modeling 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, analyzing 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) modeling environments, where the designer can smoothly maintain collections of consistent artifacts; efficiently clustering, quickly locating and reusing them, and composing different transformations.

More recently, several initiatives, including Visual Studio Code (https://code.visualstudio.com), Eclipse Che (https://www.eclipse.org/che/), Theia (https://theia-ide.org), and others, have demonstrated great promise to shift modeling 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 the state of the art suggests that versioning tools will soon be part of the modeling environment, alongside collaborative-modeling possibilities.

In Figure 1, a visual summary of this article is provided. It captures what we discussed at a single glance: how modeling 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 cyberphysical systems domains, new areas of applications have 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 have showed that modeling positively affects both engineers' productivity and the products' resulting quality, with thanks to consortia and standardization bodies.

The urge for improved platforms and foundations is stringent because of the pervasive adoption of models and related environments. This article presented some of the most daunting challenges facing a move toward a community road map. Such a road map aims to provide a motivated collection of challenges, addressing which can be used to improve modeling technology and which can be utilized to leverage adjacent research and development fields.

### Acknowledgments

The authors thank all the participants of WMM2020 (http://eventmall.info/WMM2020/), especially those who actively contributed to discussing these issues and describing them. The corresponding author of this article is Antonio Bucchiarone.

### References

1. M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*. Vol. 1. San Rafael, CA: Morgan & Claypool, 2012. [Online]. Available: https://doi.org/10.2200/S00441ED1V01Y201208SWE001
2. 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," *Softw. Syst. Model.*, vol. 17, no. 1, pp. 91–113, 2018. doi: 10.1007/s10270-016-0523-3.
3. R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Proc. Future Softw. Eng. (FOSE '07)*, May 2007, pp. 37–54. doi: 10.1109/FOSE.2007.14.
4. R. D. Caballar, "Programming without code: The rise of no-code software development," *IEEE Spectrum*, Mar. 11, 2020. [Online]. Available: https://spectrum.ieee.org/tech-talk/computing/software/programming-without-code-no-code-software-development

5. M. Tisi et al., "Lowcomote: Training the next generation of experts in scalable low-code engineering platforms," in *STAF 2019 Co-Located Events Joint Proc.*, July 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02363416
6. H. Störrle, "How are conceptual models used in industrial software development?: A descriptive survey," in *Proc. 21st Int. Conf. Eval. Assessment Softw. Eng., EASE 2017*, 2017, pp. 160–169. doi: 10.1145/3084226.3084256.
7. S. Baltes and S. Diehl, "Sketches and diagrams in practice," in *Proc. 22nd ACM SIGSOFT Int. Symp. Found. Softw. Eng. (FSE-22)*, 2014, pp. 530–541. doi: 10.1145/2635868.2635891.
8. 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.
9. 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, Switzerland: Springer-Verlag, 2018, pp. 154–160.
10. A. Dyck, A. Ganser, and H. Lichter, "A framework for model recommenders requirements, architecture and tool support," in *Proc. 2nd Int. Conf. Model-Driven Eng. Softw. Dev. (MODELSWARD)*, 2014, pp. 282–290.
11. P. J. Mosterman and H. Vangheluwe, "Computer automated multi-paradigm modeling: An introduction," *Simulation*, vol. 80, no. 9, pp. 433–450, 2004. doi: 10.1177/0037549704050532.
12. E. Rios, T. Bozheva, A. Bediaga, and N. Guilloreau, "MDD maturity model: A roadmap for introducing model-driven development," in *Proc. 2nd Euro. Conf. Model Driven Archit. — Found. Appl. (ECMDA-FA)*, 2006, pp. 78–89.
13. 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.
14. A. M. Madni and M. Sievers, "Model-based systems engineering: Motivation, current status, and research opportunities," *Syst. Eng.*, vol. 21, no. 3, pp. 172–190, 2018. doi: 10.1002/sys.21438.
15. R. L. Totterdale, "Case study: The utilization of low-code development technology to support research data collection," *Issues Inform. Syst.*, vol. 19, no. 2, pp. 132–139, 2018.

# Erratum

In the article "Benchmarking Deep Neural Network Inference Performance on Serverless Environments With MLPerf,"[1] which was published in the January/February 2021 issue of *IEEE Software*, there was an error introduced during the production process.

*Digital Object Identifier 10.1109/MS.2021.3052061*
*Date of current version: 11 February 2021*

The expansion of the acronym IE was incorrectly given as "interference engine." The correct expansion of IE is "inference engine." This is of crucial relevance in the article because it means the "inference of the deep neural network."

We sincerely apologize for this error and regret any confusion it may have caused.

## Reference

1. U. Elordi, L. Unzueta, J. Goenetxea, S. Sanchez-Carballido, I. Arganda-Carreras, and O. Otaegui, "Benchmarking deep neural network inference performance on serverless environments with MLPerf," *IEEE Softw.*, vol. 38, no. 1, pp. 81–87, Jan./Feb. 2021. doi: 10.1109/MS.2020.3030199.