Digital Twins for Software Engineering Processes

Robin Kimmel[1], Judith Michael[2], Andreas Wortmann[1], Jingxi Zhang[1]
[1] *ISW, University of Stuttgart, Stuttgart, Germany*, firstname.lastname@isw.uni-stuttgart.de
[2] *SE, RWTH Aachen University, Aachen, Germany*, michael@se-rwth.de

***Abstract*—Digital twins promise a better understanding and use of complex systems. To this end, they represent these systems at their runtime and may interact with them to control their processes. Software engineering is a wicked challenge in which stakeholders from many domains collaborate to produce software artifacts together. In the presence of skilled software engineer shortage, our vision is to leverage DTs as means for better representing, understanding, and optimizing software engineering processes to (i) enable software experts making the best use of their time and (ii) support domain experts in producing high-quality software. This paper outlines why this would be beneficial, what such a digital twin could look like, and what is missing for realizing and deploying software engineering digital twins.**

***Index Terms*—Software Engineering, Digital Twins, Artificial Intelligence**

I. Motivation

The shortage [1] of skilled workers in software engineering is a growing concern with significant implications for all industries depending on software. As the amount of software around us and the demand for novel software solutions continues to rise, the growth of the workforce of qualified software engineers is not keeping pace. With fewer skilled engineers available, companies struggle to develop new and advanced software solutions, and the pace of innovation slows [2], potentially leading to stagnation in technological advancements. This gap between required and provided software engineering capacities prevents growth and stifles innovation.

Computer science has always been a story of increasing abstraction and improving automation. Automation of software engineering processes can help bridge the gap caused by the shortage of skilled workers by taking over well-understood, mundane, repetitive tasks, allowing software engineers to focus on more complex and creative challenges. Moreover, the rise of AI-support for all phases of the software development and operations (DevOps) life-cycle suggests that an increasingly comprehensive digital representation of relevant aspects of the DevOps activities becomes feasible and can be exploited to (support) automated decision-making about the process. Such a representation could be considered a digital twin (DT) of a software engineering process.

Based on the understanding of DTs based on the data flow model of DTs [3] and the 5D model of DT components [4], we envision that such DTs, which can sense data from the represented system (*i.e.,* a software engineering process), reason about this data using internal models, and act on the represented system (*e.g.,* to suggest code improvements) will become holistic software engineering co-pilots that encompass the complete DevOps cycle instead of living in the IDE only.

Next, Sec. II illustrates what we understand as DTs, before Sec. III outlines the idea of DTs of software engineering processes. Afterward, Sec. IV illustrates an example application before Sec. V discusses our idea. Then, Sec. VI lays out our next steps toward this idea and Sec. VII concludes.

II. Background

Digital Twins: The most prominent definitions of DTs (a) separate them from digital models and digital shadows by requiring automated data flows from the original system (to sense changes in it) and to the original system (to induce changes to it) [3] and (b) require that they comprise five dimensions [4]: (1) an interface to the original system, (2) data from and about that system, (3) models representing (parts of) that system, (4) services producing added value based on data and models, and (5) connections between all of these.

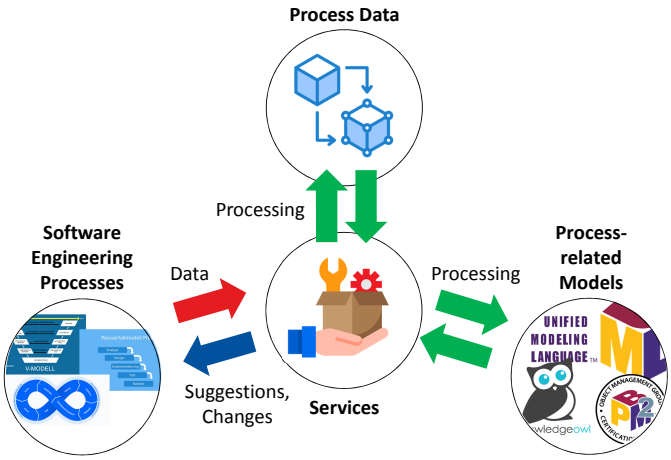

Fig. 1: 5D DTs applied to software processes [4]

Consequently, a DT is a software system that connects to the original system, observes data from it, uses its models about that system and the services to reason about these observations, and suggests changes to the system– which can be software themselves [5], [6], [7] And while there is a tremendous body of work on DTs, such as mapping studies on their nature [8], analyses of their characteristics [9], surveys on their reference architectures [10], systematic literature reviews on the development of DT software in application areas [11] and plethora of publications on their application to specific challenges, a DTs of software engineering processes is missing.

Automated Software Engineering Pipelines: There are many for automating software engineering, such as SonarQube

[12], GitLab's continuous integration and continuous delivery (CI/CD) [13], or Atlassian Bamboo [14], which primarily are used to analyze the code base and only after committing changes to the code base to the server. Some are integrated into comprehensive tool suites, where planning can be conducted through issue boards and the testing and deployment of software can be automated through the CI/CD pipeline. Although these tools facilitate the tracking of development processes, optimization techniques, such as work splitting and the simulation of software development processes, the observation of the running software products and their links to the development artifacts (not limited to the code base) are still largely underexplored. While there is ongoing research that investigates leveraging DTs on the operations side of DevOps [15], a holistic DT of software engineering processes that encompasses all relevant artifacts and engineering phases does not exist yet.

III. Dimensions of Software Process Twins

With software process DTs, we suggest transferring a part of tool and automation control from stakeholders to a comprehensive DT that (a) encompasses, and relates, all relevant activities of a specific software engineering process, and (b) actively suggests or realizes changes to the process and its artifacts instead of reacting on stakeholder inputs only.

According to the 5D model of DTs (*cf.* Fig. 1), such twins need (1) an interface to obtain data from the development process, (2) data from that process, (3) models about the process (the *knowledge*), (4) services computing changes and suggestions (the *reasoning*) on the process and its artifacts, and (5) connections between all of them. Therefore, we assume that the services are units of computation such as twins and that they connect data obtained via the interface to the process to internal models about it before producing actions regarding the process. The following outlines these dimensions for a DT for software engineering processes.

Interfaces to the process come in many forms and can include anything digitally observable in current and future SE processes. As of today, this includes data from (1) project management tools (such as issue tickets, merge requests, *etc.*), (2) interactions with CI/CD tools, co-Pilots, test results, (3) observing IDEs and the artifacts (code, dependency management, debugging information) being manipulated therein. Aside from these obvious data sources, further data might support a better understanding while improving the SE process as well, such as (4) information about the availability of developers, *e.g.,* from their work calendars (to automatically plan the distribution of tasks and monitor workload conflicts), or (5) conversations, such as meeting protocols or chat logs, about the current project and related projects (to identify topics that have been problematic in the past and might raise issues again),

Models of a software engineering process include process models that describe graphs of activities and relate these to involved stakeholders, data models about development information (e.g., tickets, properties of developers), constraint models (e.g., about available resources) that the services of the DTs use, as well as AI models, to make decisions about the process. In addition, models created within a software engineering process should be considered, e.g., architecture design models, goal models for specifying system requirements from the user perspective, sequence models specifying the system behavior, or test models describing test cases and test data.

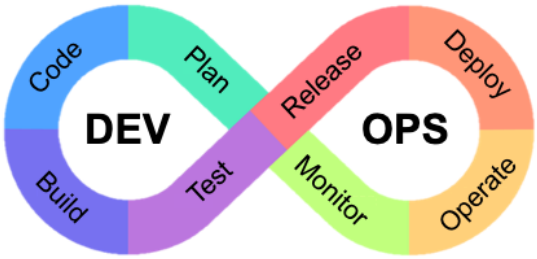

Fig. 2: The eight phases of DevOps

Based on the data and models, services can represent the state of the process, as well as analyze it and propose change suggestions. Various technologies for such services already exist and can become part of the software process DTs easily. Table 1 lists examples of such services for the different software engineering activities based on the DevOps process model (cf. Fig. 2). These activities are partially aligned with the DevOps cycle to encompass both the development and the operations phases. However, as the underlying process model, any other model, for example, the V-Model and the Waterfall model, can be used. For each service, we summarize inputs (data and models), its reasoning activities, and potential outputs.

With such data from the engineering tools, models about the process, stakeholders, and intended software product, and services in place, a comprehensive DT can support software engineering processes across the complete software lifecycle. The next section illustrates such a DT focusing on the creation of sustainable software.

IV. Example Application

When creating a DT for software engineering, we must first define the purpose for which we aim to develop it. Each of the purposes determines, which *data* is needed from which tooling in the software engineering process, which *process-related models* should be reused or have to be developed, which *services* should be developed to fulfill the purpose, e.g., analysis, optimization, prediction of specific aspects of the software and its related processes, and which *visualizations and interaction possibilities* are needed for human users of this DT. Regarding the sustainability of software systems, a similar approach has analyzed which data, models, and services could be of interest when aiming to perform a sustainability analysis of a software system as a purpose [30]. To realize such a system, a domain and purpose-tailored specification of data, models, and services is needed, as we show in the following two examples.

Imagine creating a DT of a software engineering process with the purpose of *analyzing the adherence to the*

Tbl. 1: Service input, processing, and output for selected software engineering twin activities, based on different DevOps phases [16]. The comprehensive list can be found here [17]

Phase	Service Input (Data & Models)	Service Reasoning	Service Output (Suggestions, Changes)
Plan	Open issues and successfully closed issues.	Calculation of similarity between issues closed by Developer X and currently open issues [18] [19].	Proposed ticket assignments based on similar, successfully resolved tickets from the past.
Plan	Newly created requirements (Frontend or Backend).	Adaptation of requirements to project standard with developer specific views.	Output of requirements in various formats such as natural language, UML, etc. with user specific preferences. Include recommendations for code snippets, UI layouts etc.
Code	Declarative knowledge about the project, for example, facts about existing race conditions (two or more threads accessing a specific variable).	Store information and provide an interface for a developer to retrieve declarative knowledge while working inside an IDE [20].	Provide the developer with declarative knowledge about the project, ticket etc. or directly recommend snippets based on the knowledge.
Code	Currently created code from an IDE. Developer profile.	Comparison via embeddings with a database of code snippets [21].	Indication in the IDE whether similar code has been implemented elsewhere or how others have implemented it.
Code	A developer trying to create a new feature in an existing code base.	Using a model trained via deep learning from GitHub commits to emulate, attempts to improve readability [22].	Provide an improved version of the relevant code with better readability to allow the developer to make better decisions more easily.
Code	Currently created code from an IDE.	Extract and analyze dependencies between different artifacts [23].	Suggestions on how to structure incremental builds or for reducing coupling between components.
Build	Changes in dependency versions in build artifacts (e.g., Maven's POM).	Analysis of issues (both on GitHub and internal code base) that arose after the version upgrade. Identification of code changes that match patterns corresponding to known design patterns in the local code base.	Suggestions for automatic local code adaptations based on identified patterns in related issues. Indications of potentially affected areas based on issue patterns. Recommendations for updating other dependencies based on changes observed in issue-related code.
Build	Code and target hardware specifications.	Estimation of energy consumption on target hardware.	Estimated consumption of code sections, categorized as green, yellow, or red. Recommendations for improvements.
Test	Code changes in the IDE.	Automatic creation and execution of tests in real-time to anticipate potential errors [24] [25].	Immediate feedback for the developer.
Test	Existing code base of a microservice architecture as containerized applications.	Execution of a combination of different security scanners, such as for example docker-bench-security, nmap, terrascan [26].	Security report for the given set of applications derived from different tools.
Deploy	Code in development, models of target platform and deployments.	Analysis of compatibility of code to variants of target platforms and deployments [27].	Validity of potential code deployment or list of incompatible software parts.
Operate	Logged system operation events.	Analyze runtime behavior [28].	Suggestions on how to alter the system configuration of dynamically adaptive systems.
Operate	Current real time energy consumption.	Peak analysis or comparison to predefined energy goals.	Show times of highest energy need to readjust system configuration or create energy usage reports.
Monitor	Sustainability goals as non-functional requirements and system monitoring data.	Calculation of metrics and comparison with sustainability targets.	Report on achieved or failed sustainability goals.
Monitor	User click behavior within an already deployed UI.	Comparison with learned database of personas [29].	Suggestions for UI improvements (potential future work).

architectural specification of the system under development. Required data and models could be, e.g., a wiki documenting architectural descriptions, and architecture design models as well as the code of the current implementation. Automated services analyze the code for potential contradictions with the architecture description, e.g., logically separate components unexpectedly communicate, and provide a high-level overview about which part of the specified architecture was already realized. The visualization of the analysis results depends on whom to show them: For software architects, it might be more interesting to show a visual representation of the software architecture, e.g., as an architecture model or a graph structure, with highlights where contradictions occur and the possibility to either jump into relevant parts of the implementation or which developers to contact to discuss this deviation with. For software developers, it might be more interesting to get information directly in the IDE to not distract them from the implementation flow by navigating to another tool. Here, relevant parts of the architectural specification could be shown, e.g., in components that communicate unexpectedly with another component, with the possibility of retrieving more details.

In another example, we aim to develop a DT of a software engineering process with the purpose of better ticket planning support for developers. Required data are the assigned issues with the estimated effort until the next sprint, the incoming and confirmed new appointments in the developer's calendar and the tickets that the developer closes. Automated services calculate whether there is enough time left until the end of the sprint to process the remaining open tickets. If the free time is insufficient, the developer and/or team leader could be warned via email or again in their DT cockpit where such information is collected.

Creating dashboards for DTs can be highly automated, e.g., with generators for web applications [31]. One can generate large parts of these user interfaces based on data models describing the data to be displayed and models describing the graphical user interfaces. In addition, there exist libraries with configurable GUI components [32] to create such dashboards. An example DT dashboard for visualizing parameters in a system engineering process for wind turbines can be found in [33]. The authors describe the extraction of data from external engineering tools as well as data visualization. Similar approaches can be applied to software engineering processes.

V. Discussion

Not all actions, decisions, and changes to a SE process might be digitally observable easily or at all. This includes anything discussed in person without providing protocols for this discussion, as well as business decisions affecting the project(s) in question, such as reducing the developer time assigned to them. Also, some data sources that might be useful to optimize the overall process might not be usable due to regulatory or compliance issues (e.g., scanning chat logs for issue patterns might be in conflict with the GDPR). For this, aggregating and abstraction required information locally on the developers' computers and making transparent what is shared with the DT might improve support for that system.

As the data will be spread over different applications, mechanisms to extract relevant information from each of them in an automated way have to be developed. Here, vendor lock-in and lacking APIs to reach the information in an automated way might be an additional challenge.

Often, assistance systems that interact with users directly, must be fine-tuned to make and suggest changes as often as necessary, without bothering the users as this can lead to information fatigue, and ultimately, the users may ignore provided information or deactivate the assistance system. Likewise, information overhead must be prevented. Both of these require empirical studies on the application of suggestions and artifact changes through the DT during SE projects.

VI. Future Plans

Prototype. We are currently developing DTs for cyber-physical systems in various contexts and started developing sensors and actuators for interacting with software engineering processes. These include interfacing with GitHub in its role as a project management tool and CI/CD pipeline as well as monitoring of project schedules and developer capacities. To start experimenting with these, we plan to create an extension for Visual Studio Code that captures these to (a) suggest improvements to the and (b) make changes to code artifacts directly. We will use this as the basis for empirically investigating the benefits of different services in the SE process DTs.

Pilot studies. Our institutes comprise app. 100 people who are developing software on a daily basis in different domains and in a variety of project contexts and sizes. Consequently, we will deploy the first prototype into the working groups of our institutes to evaluate and improve them in these contexts.

Empirical evaluation. Once the improved prototype is ready for experimentation, we will investigate its usability regarding correctness, completeness, and helpfulness of suggestions together with user experience experts. These experiments will be conducted with novice developers (lab courses with students of different semesters) and experienced developers (from our research transfer networks and in large consortium projects).

Functional extension. In parallel, we plan to prioritize the development of additional services of the DT with focus groups on different phases of the DevOps lifecycle and from different domains, including at least manufacturing, robotics, and construction.

Systems engineering. Supporting engineering processes with DTs does not need to be constrained to software processes. Some, especially organizational decisions, can be supported by DTs in the same way. However, for every reasoning about the subject matter, corresponding services must exist, which, in systems engineering, might pose challenges regarding the availability of data, models, and best practices. In the future, we, therefore, plan to experiment with DT support in systems engineering projects as well. Therefore, we will leverage systems engineering projects in which we are involved with companies in automotive engineering, construction, and machine engineering.

VII. Conclusion

We presented a vision for DTs of software engineering processes that capture not only the typical data and models used today but can leverage additional data (e.g., schedules) and models (such as traces or architecture models) for a novel, holistic, representation of and action on a specific software engineering process. Such holistic software engineering DTs will significantly improve the way we create, maintain, and operate software by providing novel insights and immediate improvements by combining models of the process with data from yet untapped sources and powerful AI services. This will not only impact established practices of formally trained software professionals but especially of domain experts who create software or contribute to it by applying automated best practices and established software engineering principles to their solutions. Hence, liberating significant software engineering potential that can be applied to innovating and creating added-value despite the lack of skilled software engineers.

Ultimately, such DTs will become a competitive factor in software engineering.

Acknowledgments: The authors were partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Model-Based DevOps - 505496753. Website: https://mbdo.github.io.

The authors of the University of Stuttgart were partly funded by the Ministry of Science, Research and Arts of the Federal State of Baden-Württemberg within the InnovationsCampus Future Mobility (ICM).

The authors would like to thank the German Federal Ministry for Economic Affairs and Climate Action (BMWK) for partly funding this work via the joint project: Factory-X. Website: https://factory-x.org/de/

References

[1] Financial Times, "Technology and the skills shortage," https://www.ft.com/content/b1b710a1-6d12-43e5-8508-ae4584a7289a, accessed: 2024-09-30.

[2] Forbes, "Analyzing the software engineer shortage," https://www.forbes.com/councils/forbestechcouncil/2021/04/13/analyzing-the-software-engineer-shortage, accessed: 2024-10-02.

[3] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *Ifac-PapersOnline*, 2018.

[4] F. Tao, H. Zhang, A. Liu, and A. Y. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2018.

[5] Z. Lai, D. Yuan, H. Chen, Y. Zhang, and W. Bao, "Wirelessdt: A digital twin platform for real-time evaluation of wireless software applications," in *IEEE/ACM 45th Int. Conf. on Software Engineering: Companion Proc. (ICSE-Companion)*. IEEE, 2023, pp. 146–150.

[6] J. Heluany, A. Amro, V. Gkioulos, and S. Katsikas, "Interplay of digital twins and cyber deception: Unraveling paths for technological advancements," in *ACM/IEEE 4th Int. WS on Engineering and Cybersecurity of Critical Systems (EnCyCriS) and IEEE/ACM 2nd Int. WS on Software Vulnerability*, 2024, pp. 20–28.

[7] J. Ahlgren, K. Bojarczuk, S. Drossopoulou, I. Dvortsova, J. George, N. Gucevska, M. Harman, M. Lomeli, S. M. Lucas, E. Meijer *et al.*, "Facebook's cyber–cyber and cyber–physical digital twins," in *Proceedings of the 25th International Conference on Evaluation and Assessment in Software Engineering*, 2021, pp. 1–9.

[8] M. Dalibor, N. Jansen, B. Rumpe, D. Schmalzing, L. Wachtmeister, M. Wimmer, and A. Wortmann, "A cross-domain systematic mapping study on software engineering for digital twins," *Journal of Systems and Software*, 2022.

[9] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks, "Characterising the digital twin: A systematic literature review," *CIRP journal of manufacturing science and technology*, 2020.

[10] E. Ferko, A. Bucaioni, P. Pelliccione, and M. Behnam, "Standardisation in digital twin architectures in manufacturing," in *IEEE 20th Int. Conf. on Software Architecture (ICSA)*. IEEE, 2023, pp. 70–81.

[11] M. A. Guinea-Cabrera and J. A. Holgado-Terriza, "Digital twins in software engineering—a systematic literature review and vision," *Applied Sciences*, vol. 14, no. 3, p. 977, 2024.

[12] S. SA., "Code Quality, Security & Static Analysis Tool with SonarQube | Sonar," https://www.sonarsource.com/products/sonarqube/, [Online; accessed 08-October-2024].

[13] GitLab, "Get started with GitLab CI/CD," https://docs.gitlab.com/ee/ci/, [Online; accessed 08-October-2024].

[14] Atlassian, "Bamboo: Continuous Integration und Deployment," https://www.atlassian.com/de/software/bamboo, [Online; accessed 08-October-2024].

[15] B. Combemale, J.-M. Jézéquel, Q. Perez, D. Vojtisek, N. Jansen, J. Michael, F. Rademacher, B. Rumpe, A. Wortmann, and J. Zhang, "Model-based devops: Foundations and challenges," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 2023, pp. 429–433.

[16] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, "Devops," *IEEE Software*, vol. 33, no. 3, pp. 94–100, 2016.

[17] Anonymous, "List of activities for digital twin software engineering," https://zenodo.org/records/13927534, [Online; accessed 08-October-2024].

[18] M. Z. Tunio, H. Luo, C. Wang, F. Zhao, W. Shao, and Z. H. Pathan, "Crowdsourcing software development: Task assignment using pddl artificial intelligence planning." *Journal of Information Processing Systems*, vol. 14, no. 1, 2018.

[19] H. K. Dam, T. Tran, J. Grundy, A. Ghose, and Y. Kamei, "Towards effective AI-powered agile project management," in *IEEE/ACM 41st Int. Conf on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE, 2019, pp. 41–44.

[20] M. Leung and G. Murphy, "On automated assistants for software development: The role of llms," in *38th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 1737–1741.

[21] J. Bader, S. S. Kim, F. S. Luan, S. Chandra, and E. Meijer, "Ai in software engineering at facebook," *IEEE Software*, vol. 38, no. 4, pp. 52–61, 2021.

[22] A. Vitale, V. Piantadosi, S. Scalabrino, and R. Oliveto, "Using deep learning to automatically improve code readability," in *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2023, pp. 573–584.

[23] T. Greifenberg, S. Hillemacher, and B. Rumpe, *Towards a Sustainable Artifact Model: Artifacts in Generator-Based Model-Driven Projects*. Shaker, 2017.

[24] S. Reddy, C. Lemieux, R. Padhye, and K. Sen, "Quickly generating diverse valid test inputs with reinforcement learning," in *ACM/IEEE 42nd Int. Conf. on Software Engineering*, 2020, pp. 1410–1421.

[25] S. Fatima, T. A. Ghaleb, and L. Briand, "Flakify: A black-box, language model-based predictor for flaky tests," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1912–1927, 2022.

[26] B. Ünver and R. Britto, "Automatic detection of security deficiencies and refactoring advises for microservices," in *2023 IEEE/ACM International Conference on Software and System Processes (ICSSP)*. IEEE, 2023, pp. 25–34.

[27] J. C. Kirchhof, A. Kleiss, B. Rumpe, D. Schmalzing, P. Schneider, and A. Wortmann, "Model-driven self-adaptive deployment of internet of things applications with automated modification proposals," *ACM Transactions on Internet of Things*, vol. 3, no. 4, pp. 1–30, 2022.

[28] R. Bhagwan, S. Mehta, A. Radhakrishna, and S. Garg, "Learning patterns in configuration," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 817–828.

[29] X. Zhang, H.-F. Brown, and A. Shankar, "Data-driven personas: Constructing archetypal users with clickstreams and user telemetry," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2016, pp. 5350–5359.

[30] M. Heithoff, A. Hellwig, J. Michael, and B. Rumpe, "Digital Twins for Sustainable Software Systems," in *GREENS 2023*. IEEE, July 2023, pp. 19–23.

[31] C. Buschhaus, A. Gerasimov, J. Kirchhof, J. Michael, L. Netz, B. Rumpe, and S. Stüber, "Lessons learned from applying model-driven engineering in 5 domains: The success story of the MontiGem generator framework," *Journal Science of Computer Programming*, vol. 232, p. 103033, Jan 2024.

[32] A. Gerasimov, N. Jansen, J. Michael, and B. Rumpe, "Applying self-extension mechanism to DSLs for establishing model libraries," in *23rd ACM SIGPLAN Int. Conference on Generative Programming: Concepts and Experiences (GPCE 24)*. ACM, October 2024.

[33] J. Michael, I. Nachmann, L. Netz, B. Rumpe, and S. Stüber, "Generating Digital Twin Cockpits for Parameter Management in the Engineering of Wind Turbines," in *Modellierung 2022*, ser. LNI. GI, June 2022, pp. 33–48.