[HHMR23] M. Heithoff, A. Hellwig, J. Michael, B. Rumpe:
Digital Twins for Sustainable Software Systems.
In: IEEE/ACM 7th Int. Workshop on Green And Sustainable Software (GREENS),
P. Lago, R. Kazman (Eds.), pp. 19–23, IEEE, Jul. 2023.

Digital Twins for Sustainable Software Systems

Malte Heithoff
Software Engineering
RWTH Aachen University
Aachen, Germany
heithoff@se-rwth.de

Alexander Hellwig
Software Engineering
RWTH Aachen University
Aachen, Germany
hellwig@se-rwth.de

Judith Michael
Software Engineering
RWTH Aachen University
Aachen, Germany
michael@se-rwth.de

Bernhard Rumpe
Software Engineering
RWTH Aachen University
Aachen, Germany
rumpe@se-rwth.de

***Abstract*—Sustainable software systems aim to create resource-efficient software products and reduce the carbon impact of applications. Current approaches for sustainability assessment of software are either only focused on their operation or rely on methods in their engineering. More holistic approaches for sustainable software system spanning are missing. Thus, we are interested in the engineering of sustainable software systems together with the monitoring of their sustainability goals over their whole lifetime. Within this paper, we suggest using digital twins to accompany software systems in all life cycle phases with a specific focus on using model-driven engineering methods for the creation of applications. We can generate accompanying digital twins which share relevant models and data with the actual system and provide services for the assessment of sustainability indicators. In the long run, this provides us with better assessment options for software systems.**

***Index Terms*—Model-Driven Engineering, Digital Twins, Sustainable Software Systems**

I. Introduction

When technical developments are considered in terms of their social, economic, and environmental aspects of sustainability [1], they should have a positive impact on our world. To assess this impact, the United Nations have developed 17 sustainable development goals (SDGs) with 169 associated targets [2] we should achieve. Assessing software systems [3] based on these targets requires manual effort as one has to evaluate various aspects and take data from heterogeneous data sources into account. Up to now, sustainability assessment of software systems is often a manual task. One has to manually assess different sustainability criteria [4], e.g., with scenario-based techniques [5], and continuously update the assessment in case of changes in the software.

Our aim is to investigate how to create sustainable software systems with Model-Driven Engineering (MDE) methods and monitor the sustainability goals of these synthesized systems.

We suggest using Digital Twins (DTs) to accompany software systems in all life cycle phases to reach this goal. Up to now, digital twins are mainly used to accompany Cyber-Physical Systems (CPSs), e.g., airplanes [6], cars [7], wind turbines [8], machine elements [9], injection molding machines [10], or buildings [11]. The experiences made at DT engineering for CPSs [12] can be transferred to DTs for software systems created using MDE methods. We discuss the life cycle of software systems, relevant aspects for sustainability assessment, and how MDE methods support the engineering of their DTs.

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2023 Internet of Production - 390621612. Website: https://www.iop.rwth-aachen.de

The paper is structured the following: Section II provides foundations and related work. Section III presents our vision on how to use digital twins for sustainability assessment and discusses it, and the last section concludes.

II. Foundations and Related Work

Whereas the General Assembly of the United Nations provides us with concrete 17 SDGs with 169 associated targets [2], translating these goals to software systems is still a challenge. Penzenstadler [13] defines sustainability "as preserving the function of a system over a defined time span" requiring to define the three variables system, function, and time. These can be defined in software engineering from four perspectives:

- Development processes: This includes software engineering processes with responsible use of ecological, human, and financial resources.
- Software maintenance: This includes the maintenance and evolvement of a software system with minimized environmental impact, well-managed knowledge, and sufficient economic balance.
- System production: In this perspective, the software is considered a concrete product including its hardware and the resources needed for production.
- System usage: Here, we take the entire period of use of the software and its operational environment into account.

There exists a large variety of metrics to assess green software [14]. Venters et al. [3] suggest considering software sustainability as a non-functional requirement. Measuring the extensibility, interoperability, maintainability, portability, reusability, scalability, and usability of a system enables us to make statements about its sustainability. This allows analyzing, evaluating, and reasoning about sustainability at an architectural level [15]. Kern et al. [4] describe causal chains from software products to their impacts on natural resources, e.g., energy. Design choices in software engineering, e.g., which programming language to use, compiler optimization, and implementation choices, have an influence on the energy efficiency of programs [16].

Digital Twins. We suggest using DTs to accompany software systems in all life cycle phases to support their sustainability

goals. Thus, we need to understand what constitutes a digital twin. In our understanding, *a digital twin of a system consists of a set of models of the system and a set of digital shadows, both of which are purposefully updated on a regular basis, and provides a set of services to use both purposefully with respect to the original system. The digital twin interacts with the original system by providing useful information about the system's context and sending it control commands.* [1] *Models* include several types in different languages, e.g., data, process, event, system, simulation, optimization, or 3D models. *Digital Shadows* (DSs) [17] are contextual data, and their aggregation and abstraction. In this way, we can manage the large amount of data that is captured about systems in real life by reducing it to the data we need for a specific purpose. *Services* provide the main DT functionalities and operate on models and data.

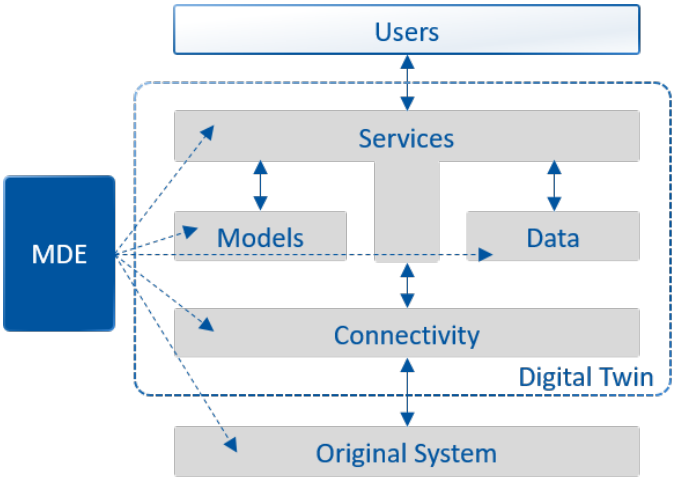

Fig. 1. MDE supports the engineering of different parts of a digital twin

Figure 1 gives an overview of where MDE can be used to develop a DT. Using MDE, we can generate software parts of the original system such as the control software of a CPS. We can generate connections between services of the DT and the original system, e.g., connect a DT visualization service with IoT devices as a part of the original system [18]. We can generate data structures as well as the connection to get data from these data sources [19]. We can apply model-to-model transformations between models to be used in a DT, e.g., extract data models from SysML models, functional models from STEP models [20], or GUI models from BPMN models [21]. Moreover, we can synthesize code for (parts of) services, e.g., for visualization [10] or connections to process mining services [22].

To the best of our knowledge, there exists no research suggesting to use DTs for supporting sustainable software systems. Recent literature suggests the use of DTs to assess sustainability targets especially for buildings, e.g., educational buildings [11], railway station buildings [23], or smart campuses [24]. Other research relies on the simulation and optimization services of DTs and investigates how to improve the sustainability performance of whole value chains, e.g., in production [25], [26], or suggest life cycle sustainability assessment in the clothing industry [27]. The main functionalities DTs provide for sustainability assessment are to monitor, calculate, and visualize sustainability indicators, and simulate and forecast these indicators based on historic information. The services DTs provide assist with responsible consumption and use of resources in relation to created products. They enable the simulation of different variants of DTs before building the physical one to improve resource efficiency. They facilitate the optimization of production processes towards waste reduction and energy saving and are allowing responsible production. Moreover, DTs provide services for self-adaptability to improve resource efficiency. Clearly, these services can also help to support sustainability goals of software systems.

[1]We published the first version of this definition in [10] but updated it to version 2.1 after discussions on a Dagstuhl seminar on DTs and further meetings, such as Modellierung'23.

III. A Vision Towards Digital Twins for Sustainable Software Systems

Our vision is that a DT accompanies software systems during their whole life cycle to enable the sustainability assessment of a software system (see Figure 2). To discuss sustainability and DTs requires taking four perspectives [13]: To design and engineer software in a sustainable way, to produce a software system in a sustainable way, to use the software in a sustainable way, and to maintain up to replace or reuse software in a sustainable way. By using a DT accompanying the whole life cycle of a software system, all four perspectives could be combined.

Software engineering produces a large amount of data, either directly as run-time data or indirectly as information gathered about the system. The DT contains models that it mostly shares with the software system. Engineering models from the design and requirements phase do not only describe the software system but also contain valuable information for the DT at its run-time, e.g., for analysis purposes, or for its own development, e.g., to describe a shared data structure [28]. Run-time models across the life cycle can be utilized to analyze current or planned behavior. The digital twin's digital shadows aggregate potentially large data from the software system, about it, or within its context and describe the information relevant for targeted purposes. The different services in a DT utilize the DSs to analyze, report, interact with, and intervene in the software system.

In the following, we examine the different life cycle phases, namely *Design*, *Implementation/ Generation*, *Operation*, and *End-of-Life* (see Figure 2) and elaborate on sustainable influences of the DT on the software system.

Design. When a system is designed using MDE, several modeling languages are used [29]: to build a data model of the domain, e.g., via UML class diagrams, to model behavior of single components, e.g., with statecharts or the BPMN, or to describe the interplay of multiple components, e.g., using an architecture description language or SysML. Often desirable or forbidden situations are modeled with exemplary methods,

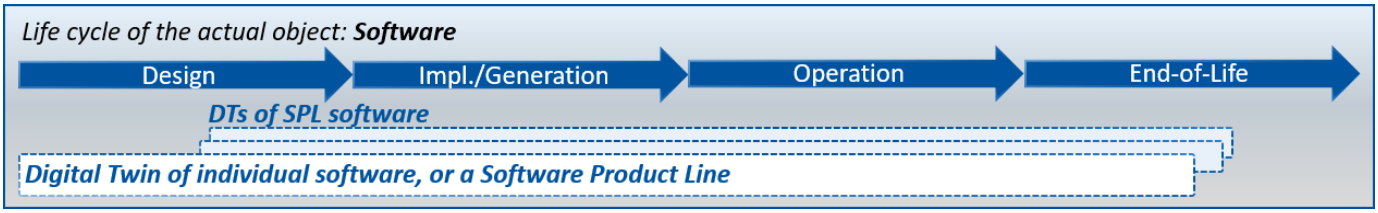

Fig. 2. Different variants of digital twins of software systems

like object or sequence diagrams. During this phase of the software life cycle, the models them-self can be considered data of the DT and can be used to provide services to measure and improve the sustainability of the software system.

By adding the system requirements to the DT, automatic requirements tracing techniques, e.g., [30], can be used to ensure that all requirements are met before deployment. This increases correctness and availability. Additionally, if every part of a system can be traced back to its original requirement, unneeded components can be identified. If the sustainability goals of the system are expressed as non-functional requirements, the DT can provide a service to monitor compliance during the whole life cycle by comparing with measurements added as a digital shadow in the later phases.

By analyzing the architecture models of the system in several situations the consumed resources can be estimated and optimized. If the usage is expected to fluctuate during the operation phase, several architectures can be simulated and compared with regard to horizontal scaling, favoring low baseline resource consumption and appropriate use of load balancing techniques. Further, scenario-based reliability analysis of the architecture, e.g., [31], can be used to estimate the impact of failures on the availability of the software system. These estimates can then be used to reach the desired availability with optimal resource usage by reducing excessive redundantly deployed components and services.

If the behavior of system components is modeled, e.g., with sequence diagrams [32], the resource usage can also be estimated with scenario-based analysis, via heuristics or simulation. Similarly to the analysis of the architecture models, the most sustainable alternative can be chosen from multiple behavior models meeting the requirements. The resource-intensive parts of the system can also be identified and optimized early on in the development. All of the sustainability estimates made in the design phase can also be considered a digital shadow of the system and can be used in later life cycle stages for further optimization.

Implementation/ Generation. In the implementation phase, hand-written code is added to the generated code. Automated tests are executed on (parts of) the system, during which logs of execution sequences and resource usage can be collected. Both, the logs and source files, can be considered in DSs of the system. Thus, the meta-model of the used general-purpose programming language(s) and a meta-model for the run time logs of the system can be added to the DT model.

Model-based testing techniques can reuse the exemplary models from the digital shadow of the design phase to ensure that the implementation conforms to the specification. For this purpose, the logs recorded during the execution of tests can be analyzed to confirm that no forbidden messages, as defined in sequence diagrams, are sent between components. Since the conformance to these parts of the specification does not need to be checked by additional handwritten tests, fewer development resources are used. Additionally, inconsistencies are found early in the development process, which reduces the resources wasted during the operation of a faulty system, as well as the development resources used to fix bugs.

By analyzing the resource usage logged during test execution, resource-intensive sections of the code can be identified and optimized before the system is experiencing most of its usage in the operation phase. The resource consumption estimates contained in the DT from the design phase can be compared to the resource usage under testing conditions. The results can be used to grasp the quality of the used analysis techniques and improve used heuristics. This improves the overall process in later development iterations if the system is built using agile methods or for further projects.

The DT should not only be used to develop sustainable software but also to develop software sustainably. Process mining can be used to discover a process model from log data [33]. By recording log data of all development tools and discovering the underlying process model using process mining, several contributions to the sustainability of a system can be made. The process model can be used to identify the least sustainable parts of the development process, e.g., by calculating the overall resource usage for each step. This allows applying optimizations where they have the highest impact. Additionally, bottlenecks in the process can be found and the tooling can be adapted to address these shortcomings, decreasing the idle time of project members.

Artifacts are in complicated relationships in most software projects, like inheritance structures in object-oriented code, and in projects following MDE in particular, since models are used for multiple different purposes. They can either be used for analysis, in model-to-model transformation or fed into a code generator, which uses templates to synthesize code. [34] describes an artifact model that can be used to analyze the hierarchical dependencies between all artifacts in MDE projects. These dependencies in the DSs of the design and implementation phase can be used to make computationally expensive operations incremental. If none of the analysis inputs, like conformance checking between an object diagram

and a class diagram, changed since the last execution, the new calculation is not needed reducing resource usage. The same principle applies to other resource-intensive parts of the MDE process, like code generation, automatic testing, and formal verification with automated theorem provers.

If the system contains components realized with machine learning (ML) techniques, training ML models is also resource intensive. [35] describe an analysis technique on model-based ML systems enabling the reuse of pre-trained ML models to avoid costly retraining of networks.

Operation. During the operation phase, the original software system produces the most amount of data. This run-time data can be logs (on different detail levels), status information, or service requests. This is where the DT has the most noticeable impact. It is connected to the original system and is provided by or gathering its data. It monitors the system's status and reports to responsible departments. Additionally, the DT is provided context information not only about the original system but also about all surroundings which allows it to gather more holistic digital shadows of the system. An important requirement for this is the assignability of data to individual components or services which makes analyses in digital shadows more expressive. As is the nature of the DT, it can also interact with and intervene in the software system at run-time. It uses the information, e.g., to optimize parameters, alter the system configuration, or organize resources. There exist already papers on how to develop DTs for physical systems in a model-driven way [18], [36]. For software at operation time, the DT can be generated in a similar way.

One key concept of the DT during operation is real-time monitoring [37]. To support the system's sustainable operation, startup, and shutdown, the DT performs purposeful analyses of the current behavior and detects execution gaps. Those can be identified from previously modeled desired behavior, e.g., sequence diagrams specifying single operations, or using process models. These undesired situations contribute to higher energy consumption and might have a direct impact on more waste influenced by the software system [38].

The DT can be equipped with default services which, e.g, identify uncommon peaks in energy consumption compared to the current workload or compared to predefined energy goals. These services need data from the software and the hardware it is running on. With such a default set of services, the DT contains well-studied monitoring tools which can be applied to different systems in a similar way. Architecture models from the design phase allow this analysis to be particularly targeted and narrowed down to specific components. With generated GUI components from engineering models which track components' behavior or energy consumption, the information gets more human-processable. After the successful identification of any need to act, the DT is capable of reporting and supporting the software system in resolving the execution gaps. These actions are specifically adapted to the software system and require thoughtful design and implementation itself. Optimizing the system includes allocating resources adjusted to the current needs, reconfiguring system parameters [36], or cleanups to guarantee durability. Actions in the code base can only be done by the corresponding development team, but the DT redeploys it with matching parameters. Storing all digital shadows with their contextual data and aggregated information allows analyses in the last life cycle phase.

End-of-life. When the software system reaches its final life cycle phase and has stopped operation, it either has successfully fulfilled its purpose, needs major re-engineering, or gets replaced by another system. Given the rapid pace of technological progress, software as a whole quickly becomes outdated without major updates. These might reach deep into the software's core design decisions. Nonetheless, a few components might have performed well and are worth be carried on to the next generation.

In this life cycle phase, sustainable aspects are primarily in reuse and data storage. Disposal is not a big concern, unlike physical systems. The DT does not collect new data but is now able to combine information from all previous life cycle phases. This helps engineers determine which component can be reused and how it can be embedded in their new system, or which data is still relevant to keep. Models from design and implementation/generation specify the component's interface, how it functions, and what role it takes in an architecture. Comparing the planned behavior in the design phase with its actual behavior from operation with methods like process conformance checking [39] together with analyses on error logs allows the DT to draw conclusions about the component's relevance and reliability in future software systems. If a component meets desired energy goals can be determined with logged energy consumption. Whether a component is easy to integrate can be derived from the implementation/generation phase by analyzing test reports regarding the success rate when using this component. With this information about the software system over previous life cycle phases, development teams gain a deeper insight and can engineer more reliable, maintainable, and energy-efficient software systems.

IV. Conclusion

Within this paper, we have introduced the use of DTs of software systems to support the sustainability assessment of applications. We have discussed the life cycle of the actual object software system and shown what models, data, and services are relevant in the different phases. We discuss the engineering process, production, maintenance, and usage of the DT from a sustainability perspective, as DTs are software systems themselves. Questions one might pose are if the engineering of an additional software system, the DT, for a software system is sustainable. As the sustainability DT only collects data and models for its specific purpose, we expect that the DT will be of a much smaller size than the original system and will not scale up with the original system. However, this question can only be answered if one has already created several sustainability DTs. Another question is why the original system needs a DT at all as it is again software and the original system could take over its tasks. To ensure the separation of interests, the software system should be

developed according to its requirements. This allows different domain experts to develop the goals more efficiently, reuse DT components for similar purposes in different software systems, and targeted generation of DT components from models. Furthermore, the DT exists and evolves along with the software system's life cycle. On the one hand, this allows the DT already operating while the software is still in development (design, implementation/generation) or not operating anymore (end-of-life). The first requires the use of MDE to quickly develop a running DT. On the other hand, the separation allows the DT to connect the different life cycle phases [12], [40] enabling more holistic analyses.

REFERENCES

[1] B. Purvis, Y. Mao, and D. Robinson, "Three pillars of sustainability: in search of conceptual origins," *Sust. Science*, vol. 14, no. 3, 2019.

[2] United Nations, "Transforming our world: the 2030 agenda for sustainable development. resolution adopted by the general assembly on 25.9.2015." 2015. [Online]. Available: https://sdgs.un.org/2030agenda

[3] C. C. Venters, L. Lau, M. K. Griffiths, V. Holmes, R. R. Ward, C. Jay, C. E. Dibsdale, and J. Xu, "The blind men and the elephant: Towards an empirical evaluation framework for software sustainability," *Journal of Open Research Software*, vol. 2, no. 1, 2014.

[4] E. Kern, L. M. Hilty, A. Guldner, Y. V. Maksimov, A. Filler, J. Gröger, and S. Naumann, "Sustainable software products-towards assessment criteria for resource and energy efficiency," *Future Generation Computer Systems*, vol. 86, pp. 199–210, September 2018.

[5] H. Koziolek, "Sustainability evaluation of software architectures: A systematic review," in *ACM SIGSOFT Conf. QoSA and ACM SIGSOFT Symp. ISARCS*. ACM, 2011, p. 3–12.

[6] V. Zaccaria, M. Stenfelt, I. Aslanidou, and K. G. Kyprianidis, "Fleet monitoring and diagnostics framework based on digital twin of aero-engines," in *Turbo Expo: Power for Land, Sea, and Air*, vol. 51128. American Society of Mechanical Engineers, 2018, p. V006T05A021.

[7] V. Damjanovic-Behrendt, "A digital twin-based privacy enhancement mechanism for the automotive industry," in *Int. Conf. on Intelligent Systems (IS)*, 2018, pp. 272–279.

[8] 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*. GI, 2022, pp. 33–48.

[9] W. T. Gruender, "Systems engineering requires digital twins of machine elements," in *CONAT 2016 Int. Congress of Automotive and Transport Engineering*. Springer, 2017, pp. 227–233.

[10] M. Dalibor, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Towards a Model-Driven Architecture for Interactive Digital Twin Cockpits," in *Conceptual Modeling*. Springer, 2020, pp. 377–387.

[11] L. C. Tagliabue, F. R. Cecconi, S. Maltese, S. Rinaldi, A. L. C. Ciribini, and A. Flammini, "Leveraging digital twin for sustainability assessment of an educational building," *Sustainability*, vol. 13, no. 2, 2021.

[12] S. Fur, M. Heithoff, J. Michael, L. Netz, J. Pfeiffer, B. Rumpe, and A. Wortmann, "Sustainable Digital Twin Engineering for the Internet of Production," in *Digital Twin Driven Intelligent Systems and Emerging Metaverse*. Springer, 2023.

[13] B. Penzenstadler, "Towards a definition of sustainability in and for software engineering," in *ACM Symp. on Applied Comp. (SAC)*, 2013.

[14] P. Lago, Q. Gu, and P. Bozzelli, "A systematic literature review of green software metrics," 2014. [Online]. Available: https://research.vu.nl/en/publications/a-systematic-literature-review-of-green-software-metrics

[15] C. C. Venters, R. Capilla, S. Betz, B. Penzenstadler, T. Crick, S. Crouch, E. Y. Nakagawa, C. Becker, and C. Carrillo, "Software sustainability: Research and practice from a software architecture viewpoint," *Journal of Systems and Software*, vol. 138, pp. 174–188, 2018.

[16] S. Abdulsalam, D. Lakomski, Q. Gu, T. Jin, and Z. Zong, "Program energy efficiency: The impact of language, compiler and implementation choices," in *International Green Computing Conference*, 2014, pp. 1–6.

[17] F. Becker, P. Bibow, M. Dalibor, A. Gannouni, V. Hahn, C. Hopmann, M. Jarke, I. Koren, M. Kröger, J. Lipp, J. Maibaum, J. Michael, B. Rumpe, P. Sapel, N. Schäfer, G. J. Schmitz, G. Schuh, and A. Wortmann, "A Conceptual Model for Digital Shadows in Industry and its Application," in *Conceptual Modeling, ER 2021*. Springer, 2021.

[18] J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Model-driven Digital Twin Construction: Synthesizing the Integration of Cyber-Physical Systems with Their Information Systems," in *Int. Conf. on Model Driven Eng. Languages and Systems*. ACM, 2020.

[19] K. Adam, J. Michael, L. Netz, B. Rumpe, and S. Varga, "Enterprise Information Systems in Academia and Practice: Lessons learned from a MBSE Project," in *40 Years EMISA (EMISA'19)*, ser. LNI. GI, 2020.

[20] B. Caesar, N. Jansen, M. Weigand, M. Ramonat, C. S. Gundlach, A. Fay, and B. Rumpe, "Extracting functional machine knowledge from step files for digital twins," in *IEEE Int. Conf. on Emerging Technologies and Factory Automation (ETFA)*, 2022, pp. 1–4.

[21] D. Bano, J. Michael, B. Rumpe, S. Varga, and M. Weske, "Process-Aware Digital Twin Cockpit Synthesis from Event Logs," *Journal of Computer Languages (COLA)*, vol. 70, June 2022.

[22] T. Brockhoff, M. Heithoff, I. Koren, J. Michael, J. Pfeiffer, B. Rumpe, M. S. Uysal, W. M. P. van der Aalst, and A. Wortmann, "Process Prediction with Digital Twins," in *Int. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. ACM/IEEE, 2021.

[23] S. Kaewunruen and N. Xu, "Digital twin for sustainability evaluation of railway station buildings," *Frontiers in Built Environment*, vol. 4, 2018.

[24] A. Zaballos, A. Briones, A. Massa, P. Centelles, and V. Caballero, "A smart campus' digital twin for sustainable comfort monitoring," *Sustainability*, vol. 12, no. 21, p. 9196, 2020.

[25] A. Barni, A. Fontana, S. Menato, M. Sorlini, and L. Canetta, "Exploiting the digital twin in the assessment and optimization of sustainability performances," in *Int. Conf. on Intelligent Systems (IS)*, 2018.

[26] L. Li, T. Qu, Y. Liu, R. Y. Zhong, G. Xu, H. Sun, Y. Gao, B. Lei, C. Mao, Y. Pan, F. Wang, and C. Ma, "Sustainability assessment of intelligent manufacturing supported by digital twin," *IEEE Access*, vol. 8, 2020.

[27] T. Riedelsheimer, L. Dorfhuber, and R. Stark, "User centered development of a digital twin concept with focus on sustainability in the clothing industry," *Procedia CIRP*, vol. 90, pp. 660–665, 2020, 27th CIRP Life Cycle Engineering Conference (LCE2020).

[28] M. Dalibor, M. Heithoff, J. Michael, L. Netz, J. Pfeiffer, B. Rumpe, S. Varga, and A. Wortmann, "Generating Customized Low-Code Development Platforms for Digital Twins," *Journal of Computer Languages (COLA)*, vol. 70, June 2022.

[29] T. Clark, M. v. d. Brand, B. Combemale, and B. Rumpe, "Conceptual Model of the Globalization for Domain-Specific Languages," in *Globalizing Domain-Specific Languages*, ser. LNCS 9400. Springer, 2015.

[30] S. Haidrar, H. Bencharqui, A. Anwar, J.-M. Bruel, and O. Roudies, "Reqdl: A requirements description language to support requirements traces generation," in *IEEE Int. Requ. Eng. Conf. WSs (REW)*, 2017.

[31] B. Tekinerdogan, H. Sozer, and M. Aksit, "Software architecture reliability analysis using failure scenarios," *Journal of Systems and Software*, vol. 81, no. 4, pp. 558–575, 2008.

[32] K. Hölldobler, J. Michael, J. O. Ringert, B. Rumpe, and A. Wortmann, "Innovations in Model-based Software and Systems Engineering," *The Journal of Object Technology*, vol. 18, no. 1, pp. 1–60, 2019.

[33] W. Van Der Aalst, "Process mining," *Communications of the ACM*, vol. 55, no. 8, pp. 76–83, 2012.

[34] T. Greifenberg, S. Hillemacher, and B. Rumpe, *Towards a Sustainable Artifact Model: Artifacts in Generator-Based Model-Driven Projects*. Aachener Informatik-Berichte, Software Engineering (30). Shaker, 2017.

[35] A. Atouani, J. C. Kirchhof, E. Kusmenko, and B. Rumpe, "Artifact and Reference Models for Generative Machine Learning Frameworks and Build Systems," in *20th ACM SIGPLAN Int. Conf. on Generative Programming: Concepts and Experiences (GPCE 21)*. ACM, 2021.

[36] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, and A. Wortmann, "Model-Driven Development of a Digital Twin for Injection Molding," in *Int. Conf. on Advanced Inf. Sys. Eng. (CAiSE'20)*, ser. LNCS, vol. 12127. Springer, 2020, pp. 85–100.

[37] K.-J. Wang, Y.-H. Lee, and S. Angelica, "Digital twin design for real-time monitoring–a case study of die cutting machine," *International Journal of Production Research*, vol. 59, no. 21, pp. 6471–6485, 2021.

[38] W. M. van der Aalst, O. Hinz, and C. Weinhardt, "Sustainable systems engineering," pp. 1–6, 2023.

[39] W. Van der Aalst, A. Adriansyah, and B. van Dongen, "Replaying history on process models for conformance checking and performance analysis," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 2, pp. 182–192, 2012.

[40] T. Y. Pang, J. D. Pelaez Restrepo, C.-T. Cheng, A. Yasin, H. Lim, and M. Miletic, "Developing a digital twin and digital thread framework for an 'Industry 4.0'Shipyard," *Applied Sciences*, vol. 11, no. 3, 2021.