A Rapid Review on Software Reuse in Digital Twin Development Lifecycle

Lorena Mamede Botelho, Claudia Werner, Claudio Miceli de Farias

Programa de Engenharia de Sistemas e Computação (PESC)
Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro – RJ – Brasil

{lorenamb, werner, cmicelifarias}@cos.ufrj.br

Abstract. *Digital Twin has become a trend in literature in the past decade. Despite its popularity, researchers reported the lack of standardized development approaches and general guidelines to foster its adoption. In this study, we conducted a rapid review to investigate whether, and under what conditions, Software Reuse techniques can address these challenges. Through a content analysis of twenty-two primary studies, results suggest that Software Reuse techniques contribute to reducing development time and enhancing composability in the Digital Twin construction process. Component-Based Software Engineering emerged as the predominant reuse approach among the analyzed studies. However, the review also identified a notable lack of empirical evaluation regarding the proposed reusable assets, limiting the evidence base for the reported benefits of Software Reuse in Digital Twin development.*

1. Introduction

The concept of Digital Twin (DT) was originally introduced by [Grieves 2003], with the term itself later formalized by NASA in their draft strategic roadmap [Shafto et al. 2010]. DT can be defined as a virtual representation of a physical entity, characterized by a bidirectional data flow that enables continuous synchronization between both. There is a set of virtual operations in the virtual space defined by Grieves, such as modeling, simulation, and optimization. Although numerous domain-specific DT examples exist in the literature, such as Manufacturing, Aerospace, and Healthcare [Semeraro et al. 2021], there remains a lack of consolidated standards, tools, and development guidelines. As stated by [Autiosalo et al. 2019], the implementations presented in the literature differ from one another, being tailored to specific use cases and implemented with a wide variety of tools. This approach reduces generality and limits its transferability to new application areas; every new adoption needs to be created from scratch [Abo-Khalil 2023].

A possible solution to this challenge would be applying Software Reuse (SR) techniques within the DT development process. According to [Krueger 1992]: *"SR is the process of creating software systems from existing software rather than building software systems from scratch."*. The concept was first conceived in 1968 to address issues related to the lack of standards and systematic approaches in software development. It promotes the use and development of reusable libraries, frameworks, and design patterns, reducing development time, time to market, and costs [Shang et al. 2022]. SR aims to make software development faster, better, and cheaper, giving competitive business advantages to the software organizations [Jacobson et al. 1997]. Applying it in DT implies using existing

software artifacts during the creation of a new DT. These artifacts can be reused at different levels of abstraction and across different stages of the software development lifecycle, not limited to code reuse [Freeman 1983]. This can reduce the cognitive distance between conceptualization and implementation, streamline development, and foster the creation of domain-independent DT solutions. SR could help address the scalability and standardization challenges observed in DT projects [Abo-Khalil 2023, Muctadir et al. 2024]. However, there are few literature reviews focused on SR, and even fewer that specifically address it within the DT development. That can be explained by the challenges inherent in accurately assessing reusability. Many studies tried to accomplish this task by proposing metrics and factors, but they could only measure factors that support reusability and not reusability itself [Mehboob et al. 2021].

Given the emerging nature of DT and the lack of consolidated evidence on SR adoption within this field, there is a clear need for a focused literature synthesis. This study proposes conducting a Rapid Review (RR), an evidence synthesis method designed to provide a structured overview of emerging research topics quickly [Tricco et al. 2015]. The goal is to investigate whether SR techniques have been applied in DT development, how they have been implemented, and what benefits have been reported. As its primary contribution, this study seeks to answer the following research questions: *"Are SR techniques used in the construction of DTs?"*, *"In which stages of the software lifecycle are they applied: Design, Implementation, or Deployment?"*, *"Does SR in DT bring improvements in standardization, software quality, or time to market?"*, and *"What experimentation methods were used to validate reuse in DTs?"*.

2. Methodology

Given the accelerated growth of DT publication in recent years, there is a need to establish concise guidelines and standards that promote the quality of software and transfer knowledge of this concept. Traditional systematic literature reviews, although widely recognized within the Evidence-Based Software Engineering community [Kitchenham et al. 2007], are very time-intensive, easily requiring months to produce actionable results. Given this, the RR methodology was chosen to conduct this study. It enables the assessment of recent relevant published studies while maintaining methodological rigor. It is particularly suitable for fast-evolving research areas, where timely analysis is essential to inform both academic research and industrial practice. To support transparency and reproducibility of the study, the Preferred Reporting Items for Systematic Reviews and Meta-Analysis (PRISMA) was used to select the articles [Page et al. 2021]. The selected database was Scopus, due to its comprehensive coverage of studies. The formulation of the search string was guided by the Population, Intervention, Comparison, Outcome, Context (PICOC) framework [Kitchenham et al. 2007], as shown by Table 1.

To define the SR approaches, the taxonomies used were the ones mentioned by [Krueger 1992, Jacobson et al. 1997, Muctadir et al. 2024], such as Architecture-based reuse, Component-based Software, Software Frameworks, Model-driven Development (MDD) and Model Driven Engineering (MDE). For Context definition, the software lifecycle stages considered were derived from the Software Development Life Cycle (SDLC) definition [Shylesh 2017]. The stages were summarized in: design, implementation and deployment, to express three levels of abstraction. The search string was validated using two control studies [AboElHassan and Yacout 2023, Autiosalo et al. 2019]. The criteria

Table 1. Search Terms with PICOC method

PICOC	Terms	Queries
Population	Digital Twin	(Digital AND Twin*) OR ”Digital Replica” OR ”Digital Shadow” OR ”Virtual Counterpart” OR ”Virtual Twin”
Intervention	Software Reuse	”Software Reuse” OR ”Architecture” OR ”Component” OR ”Framework” OR ”MDD” OR ”MDE”
Comparison	-	-
Outcome	QoS, reduced time-to-market	”Guideline” OR ”General approach” OR ”General procedure” OR ”Quality of Software” OR ”Time to market” OR ”Systematic Approach” OR ”Reusability”
Context	Software lifecycle	Design* OR Develop* OR ”Implementation” OR Deploy*

focus on primary studies written in English. Scopus data statistics using the search string ”Digital Twin” indicate that DT publications grew exponentially only after 2019; therefore, only publications between 2019 and 2024 were considered. Studies were excluded if they met at least one of the following conditions: (i) they mentioned reuse without demonstrating its application to DT building; or (ii) they addressed DT without any explicit consideration of SR. Table 2 summarizes the criteria. To enable the systematic evaluation of the SR presence in a reproducible way, the coding assessment used by this review was an adaptation of Krueger's taxonomy [Krueger 1992] to the DT context: **(i) Abstraction**: Identifies the types of software artifacts being reused (e.g., architectures, components, frameworks, models) and the abstraction mechanisms used to describe them; **(ii) Selection**: Examines the strategies, tools, or criteria implemented to select suitable reusable artifacts for specific DT use cases; **(iii) Specialization**: Analyzes how reusable artifacts were adapted, configured, or extended to meet the particular requirements of the DT domain application; **(iv) Integration**: Investigates how the selected and specialized artifacts were assembled or integrated into the final DT software system.

Table 2. Inclusion and Exclusion Criteria

ID	Inclusion	Exclusion
1	Studies published between 2019 and 2024	Duplicate publications
2	Articles written in English	Studies mentioning reuse without applying it within DT lifecycle stages
3	Primary studies	Studies addressing DT without applying any software reuse technique
4	Explicitly answer at least one research question	Books and Conference proceedings

3. Analysis

Figure 1 shows the PRISMA steps. The data synthesis was structured according to the stages of SDLC to provide a clear mapping between reuse techniques and development phases. For each SDLC phase, the reuse techniques applied were grouped in categories assessed by their benefits and validation across the four dimensions of SR.

Design Phase: Most studies adopt Composition-based Software Engineering (CBSE), where DTs are modeled as modular building blocks composed in architectures [Zhu et al. 2021, Roque Rolo et al. 2021, Steindl and Kastner 2021, Purcell et al. 2022, Xu et al. 2022, Aziz et al. 2023, Amadeo et al. 2024, Gil et al. 2024], frameworks [Zhu et al. 2021, Steindl and Kastner 2021, Clausen et al. 2021,

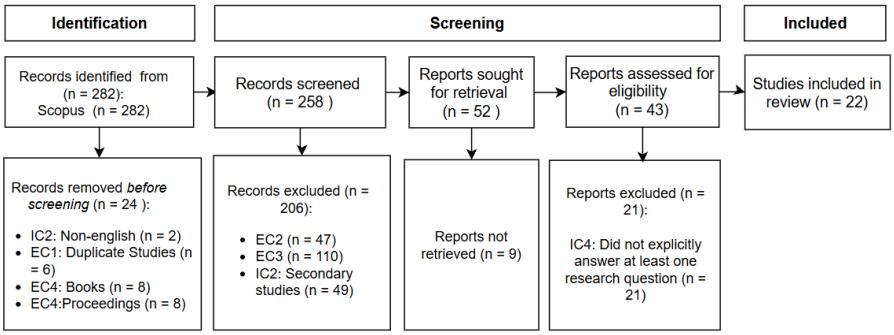

Figure 1. PRISMA diagram of selection process

Xu et al. 2022, Zhang et al. 2022, Aziz et al. 2023, Mrzyk et al. 2023, Kibira et al. 2023, Chen et al. 2024], or reference models [Bevilacqua et al. 2020]. For generic approaches, this strategy facilitates the flexibility to embrace different domains or general-purpose DTs. At the abstraction dimension, studies focused on defining these blocks upon DT characteristics, presenting them as components or services, and describing their interconnections. In [Steindl and Kastner 2021, Bevilacqua et al. 2020, Roque Rolo et al. 2021, Xu et al. 2022, Zhang et al. 2022, Amadeo et al. 2024], additional abstraction was proposed in layers that encompass environments and their elements. In all studies, the components are combined to act as a single DT, and, in some of them [Kibira et al. 2023, Amadeo et al. 2024, Latsou et al. 2024, Chen et al. 2024, Gil et al. 2024], resulting DTs can also be combined to perform as a complex system, providing high-level abstractions. Some studies have reused previously developed components, mainly for simulation and modeling features in domain-specific approaches [Purcell et al. 2022]. In [Qamsane et al. 2021], component design was done using Oriented-Object approaches.

Some of the component creation strategies adopted were oriented by the 3-dimension DT design model (developed by [Grieves 2003]), the DT 5-dimension expanded design model (mentioned by [Qi et al. 2021]), Digital twin Manufacturing Framework (defined by ISO 23247 [International Organization for Standardization 2021]), BOSS (created by [Dawson-Haggerty et al. 2013]), or SOA principles. In generic approaches, the components offer a variety of composability as a specification for each use case, which attends to the specialization dimension of SR. These compositions can be consolidated into specifications for future reuse. For domain-specific approaches, as in [Bevilacqua et al. 2020, Clausen et al. 2021, Zhu et al. 2021, Xu et al. 2022, Purcell et al. 2022, Zhang et al. 2022, Abo-Khalil 2023, Kibira et al. 2023, Chen et al. 2024], the specialization was dictated by the domain parameters. The components were designed to be ad-hoc domain features.

For integration, studies commonly adopted Service-Oriented Architectures (SOA) and Microservices Architecture principles to ensure loose coupling between components. Event-driven communication using brokers (e.g., Pub/Sub [Clausen et al. 2021, AboElHassan and Yacout 2023], MQTT [Xu et al. 2022, Aziz et al. 2023]) and REST APIs [Roque Rolo et al. 2021, Aziz et al. 2023, AboElHassan and Yacout 2023] were prevalent strategies. Data integration with third-party systems (e.g., SCADA, PLCs, Grasshopper) was also

noted in [Bevilacqua et al. 2020, Xu et al. 2022, Purcell et al. 2022]. Regarding selection, some studies proposed component discovery tools such as DT Registries [Aziz et al. 2023], DM Pools [AboElHassan and Yacout 2023], or common registries [Amadeo et al. 2024], for model, component, or service selection.

Feature-based Software Engineering (FBSE), as applied in [Autiosalo et al. 2019, Van Dinter et al. 2023, Mrzyk et al. 2023], shares similarities with CBSE by promoting modularity during the Design phase. In this approach, modules or blocks are abstractions of the product features. Features can be selected and combined to offer specialization. In all studies, domain analysis was a crucial strategy to define abstraction and specialization. In the analyzed studies, integration also occurs through distributed architectures principles, using mainly REST APIs [Autiosalo et al. 2019] and a microservices approach [Mrzyk et al. 2023]. 5C architecture for Cyber-Physical Systems [Lee et al. 2015] with Pub/Sub was shown in [Van Dinter et al. 2023]. Semantic Software Engineering (SSE) in SR context encompasses the reuse of formal semantic artifacts. In [Steindl and Kastner 2021, Roque Rolo et al. 2021, Latsou et al. 2024, Gil et al. 2024], SSE is applied using an ontology modeling technique to define entities to be implemented in the development phase, providing abstraction. In [Steindl and Kastner 2021], this process was guided by the Generic Digital Twin Architecture (presented by [Steindl et al. 2020]), which promotes shared knowledge for contextual information. In the abstraction level, [Gil et al. 2024] uses Skill-Based Engineering to define ontology entities. In the specialization dimension, each entity's attributes and abilities are specified through the Ontology Web Language. Model-driven Engineering (MDE) is an approach to raise code abstraction through high-level models; it is used to design components or modules that can be further inputs for code generation. In [Wilking et al. 2024], Model-Based System Engineering was used to support MDE, with SysML as a tool for component design. The models created are integrated along the development life cycle for validation and implementation phases.

Development Phase: Lean Digital Twin Methodology (LDTM) [Aziz et al. 2023], adapted from Lean Startup Methodology (LSM), was used to guide agile DT development, emphasizing iterative prototyping, requirement refinement, and rapid deployment. The methodology systematically prioritized the reuse of existing solutions in its *Identify Solutions and Technologies* phase, addressing the selection and specialization reuse dimensions. In Model-driven Development (MDD), models developed in the design phase (through MDE or SSE) are used to automatically generate code. This technique was found in [Wilking et al. 2024], mainly using Object Constraint Language (OCL) to expand previous models, and in [Zhang et al. 2022], using geometric models. The main benefits mentioned were the automatic instantiation of DTs tailored to any use case, promoting specialization. For some physical modeling, machine learning, and web development, algorithm libraries were used in [Zhu et al. 2021, Clausen et al. 2021, Roque Rolo et al. 2021, Xu et al. 2022, Zhang et al. 2022, AboElHassan and Yacout 2023, Mrzyk et al. 2023, Kibira et al. 2023, Zhao et al. 2023, Gil et al. 2024, Latsou et al. 2024, Chen et al. 2024].

Deployment Phase: Reuse strategies included the formalization of deployable units as Package Business Capability (PBC) [Aziz et al. 2023]. These PBCs defined deployment specifications in standardized formats (e.g., JSON), facilitating consistent in-

stantiation. A similar concept was used by [Autiosalo et al. 2019], but called Digital Twin Class (DTC). This also provided versioning for selection and attributes for personalized settings, providing specialization. SR is restricted to the design and implementation phase, and not the execution phase [Freeman 1983]. That is why the concept is not mentioned for the infrastructure environment. The emerging use of Infrastructure as Code (IaC) enables environment configuration to be treated as a reusable software artifact, promoting the use of code and other development concepts to automate and centralize deployment configuration. The current study considered it as a reuse strategy since it can provide solutions in all four reuse dimensions. In some of the CBSE approaches [Aziz et al. 2023, AboElHassan and Yacout 2023], deployment was also designed as a component, whose core responsibilities involved the configuration and orchestration of services and components. For those approaches, abstraction occurs through virtualization, such as containerization and virtual machines. In [Aziz et al. 2023, AboElHassan and Yacout 2023], the use of container images provided the reuse of the environment configuration for creating new virtualized instances. The selection of the right image is done by versioning strategies and a repository to store them. Integration was provided by [Steindl and Kastner 2021] using Business Process Model and Notation (BPMN) to establish workflows for orchestration.

Validation: The validation methods were mainly Proof of Concept (PoC) [Steindl and Kastner 2021, Purcell et al. 2022, Aziz et al. 2023] and Case Studies [Qamsane et al. 2021, Zhu et al. 2021, Roque Rolo et al. 2021, Clausen et al. 2021, Zhang et al. 2022, Abo-Khalil 2023, AboElHassan and Yacout 2023, Van Dinter et al. 2023, Zhao et al. 2023, Mrzyk et al. 2023, Latsou et al. 2024, Wilking et al. 2024, Chen et al. 2024, Gil et al. 2024]. These validations were designed to test different domain scenarios to assess the flexibility of the proposal. They covered the creation of new DTs and their absorption into the Virtual Space. Quantitative evaluations primarily assessed performance metrics such as throughput, latency, and time-to-integration [Aziz et al. 2023] or domain-specific metrics. Qualitative validation focused on software qualities like scalability and fault-tolerance [AboElHassan and Yacout 2023, Latsou et al. 2024], though in some cases, reuse-related benefits (e.g., reduced development time) were inferred rather than empirically measured.

4. Discussion

This study revealed that SR techniques are indeed being applied in DT construction, predominantly during the Design phase. Regarding the distribution of techniques regarding the stages, we found: (Design, 21) (Development, 15) (Deployment,4). It is noticeable that most techniques focus on the Design phase. That could suggest that high-level abstraction is more achievable than low-level abstractions, which may depend on specific-domain characteristics. CBSE emerged as the most frequently used strategy (Figure 2), enabling modularization and enhanced flexibility. However, there is a noticeable underrepresentation of reuse techniques in the Development and Deployment stages, suggesting a gap in the systematic application of SR throughout the DT lifecycle. The lack of diversity in reuse strategies, minimal attention to deployment-phase reuse, and limited validation metrics suggest that the DT community is still in the early stages of systematically embracing SR. In particular, few studies addressed the selection dimension of reuse, and empirical validation remains weak. Practitioners aiming to build scalable and

maintainable DTs should prioritize modular design approaches like CBSE and FBSE. The documented frameworks and models from the reviewed studies provide useful references for initiating reuse-oriented DT development. There is a clear need for further empirical studies that assess the impact of SR on DT quality attributes such as scalability, flexibility, and maintainability. Researchers are encouraged to explore reuse techniques beyond design, especially in deployment automation (e.g., IaC).

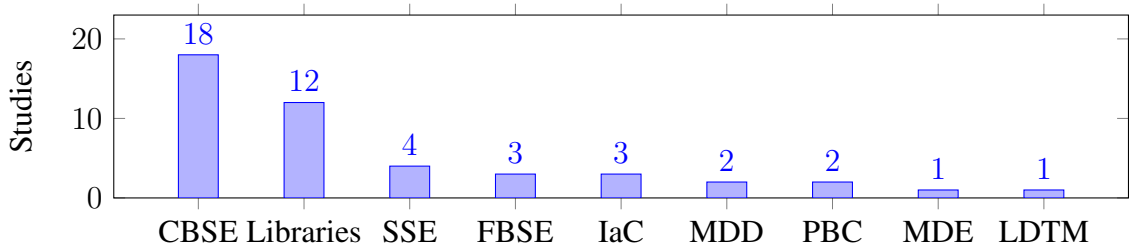

Figure 2. Distribution of Studies by Reuse Technique

5. Conclusion

This study answered the research questions proposed. For the first question, *"Are SR techniques used in the construction of DTs?"*, the answer is affirmative. SR techniques have been applied in DT construction, although their diversity remains limited. The most recurrent and dominant approach observed was CBSE. *"In which stages of the software lifecycle are they applied: Design, Implementation or Deployment?"*, the analysis revealed that SR techniques are predominantly applied during the Design phase. Some instances of reuse at the Development and Deployment stages were identified, especially with the adoption of libraries and IaC. *"Does SR in DT bring improvements in standardization, software quality, or time to market?"*, explicit discussions of reuse benefits in the reviewed literature were limited. Reported advantages focused more on composability, flexibility, and standardization of DT architectures, with less emphasis on quantitative measurements of time reduction, cost savings, or quality improvements. Most studies only implied these benefits or cited them as expected outcomes. *"What experimentation methods were used to validate reuse in DTs?"*, few formal validation methods were found in the literature that provide quantitative and qualitative metrics of reuse benefits; most of the benefits were implied or taken for granted. Aspects as reduction in cost, time to market, computational cost, and development time were merely mentioned.

Based on the studies collected and analyzed, the current trend in DT development process includes the use of CBSE as the main approach to achieve reusability. This architectural approach supports the composition of DT solutions toward more generic and scalable frameworks, enabling the incremental addition of new DT features and data providers. It facilitates the representation of complex physical systems by modularizing them into smaller, testable components, thereby reducing development complexity and enhancing fault tolerance. The formalization of these components in a generic architecture enables the application of DT in multiple domains. Although the constraints of a RR, such as a restricted database and limited time for in-depth analysis, this study can serve as a practical reference for researchers and practitioners seeking to understand and adopt the most prevalent SR techniques in DT construction. The cataloged studies provide valuable use-case examples of how these techniques have been implemented in practice. For

future work, there is a need for the development and adoption of validation methods and metrics focused on assessing the specific benefits of SR in DT projects. This would enable more informed decision-making regarding the selection of reuse techniques tailored to each use case. It is also recommended to apply reuse techniques in more phases of the development lifecycle, to provide a full-cycle reuse. In conclusion, this study emphasizes that the growing adoption and scalability of DT solutions are closely linked to the strategic application of SR techniques, which help address the main challenges in complexity management, cost reduction, and system scalability.

References

Abo-Khalil, A. G. (2023). Digital twin real-time hybrid simulation platform for power system stability. *Case Studies in Thermal Engineering*, 49:103237.

AboElHassan, A. and Yacout, S. (2023). A digital shadow framework using distributed system concepts. *Journal of Intelligent Manufacturing*, 34(8):3579–3598.

Amadeo, M., Marche, C., Ruggeri, G., Ranjbaran, S., and Nitti, M. (2024). Composing digital twins for internet of everything applications: A user-centric perspective. In *2024 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 73–78. IEEE.

Autiosalo, J., Vepsäläinen, J., Viitala, R., and Tammi, K. (2019). A feature-based framework for structuring industrial digital twins. *IEEE access*, 8:1193–1208.

Aziz, A., Chouhan, S. S., Schelén, O., and Bodin, U. (2023). Distributed digital twins as proxies-unlocking composability and flexibility for purpose-oriented digital twins. *IEEE Access*, 11:137577–137593.

Bevilacqua, M., Bottani, E., Ciarapica, F. E., Costantino, F., Di Donato, L., Ferraro, A., Mazzuto, G., Monteriù, A., Nardini, G., Ortenzi, M., et al. (2020). Digital twin reference model development to prevent operators' risk in process plants. *Sustainability*, 12(3):1088.

Chen, Z., Surendraarcharyagie, K., Granland, K., Chen, C., Xu, X., Xiong, Y., Davies, C., and Tang, Y. (2024). Service oriented digital twin for additive manufacturing process. *Journal of Manufacturing Systems*, 74:762–776.

Clausen, A., Arendt, K., Johansen, A., Sangogboye, F. C., Kjærgaard, M. B., Veje, C. T., and Jørgensen, B. N. (2021). A digital twin framework for improving energy efficiency and occupant comfort in public and commercial buildings. *Energy Informatics*, 4:1–19.

Dawson-Haggerty, S., Krioukov, A., Taneja, J., Karandikar, S., Fierro, G., Kitaev, N., and Culler, D. (2013). {BOSS}: Building operating system services. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 443–457.

Freeman, P. (1983). Reusable software engineering: Concepts and research directions. In *ITT Proceedings of the Workshop on Reusability in Programming*, volume 129, page 137.

Gil, S., Schou, C., Mikkelsen, P. H., and Larsen, P. G. (2024). Integrating skills into digital twins in cooperative systems. In *2024 IEEE/SICE International Symposium on System Integration (SII)*, pages 1124–1131. IEEE.

Grieves, M. W. (2003). Plm–beyond lean manufacturing. *Manufacturing Engineering*, 130(3):23–23.

International Organization for Standardization (2021). ISO/IEC 23247 - Digital Twin Framework for Manufacturing. `https://www.iso.org/standard/75066.html`. Part 1: Overview and General Principles. Geneva, Switzerland.

Jacobson, I., Griss, M., and Jonsson, P. (1997). *Software reuse: architecture, process and organization for business success*. ACM Press/Addison-Wesley Publishing Co.

Kibira, D., Shao, G., and Venketesh, R. (2023). Building a digital twin of an automated robot workcell. In *2023 Annual Modeling and Simulation Conference (ANNSIM)*, pages 196–207. IEEE.

Kitchenham, B., Charters, S., et al. (2007). Guidelines for performing systematic literature reviews in software engineering.

Krueger, C. W. (1992). Software reuse. *ACM Comput. Surv.*, 24(2):131–183.

Latsou, C., Ariansyah, D., Salome, L., Erkoyuncu, J. A., Sibson, J., and Dunville, J. (2024). A unified framework for digital twin development in manufacturing. *Advanced Engineering Informatics*, 62:102567.

Lee, J., Bagheri, B., and Kao, H.-A. (2015). A cyber-physical systems architecture for industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23.

Mehboob, B., Chong, C. Y., Lee, S. P., and Lim, J. M. Y. (2021). Reusability affecting factors and software metrics for reusability: A systematic literature review. *Software: Practice and Experience*, 51(6):1416–1458.

Mrzyk, P., Kubacki, J., Luttmer, J., Pluhnau, R., and Nagarajah, A. (2023). Digital twins for predictive maintenance: a case study for a flexible it-architecture. *Procedia CIRP*, 119:152–157.

Muctadir, H. M., Manrique Negrin, D. A., Gunasekaran, R., Cleophas, L., van den Brand, M., and Haverkort, B. R. (2024). Current trends in digital twin development, maintenance, and operation: An interview study. *Software and Systems Modeling*, pages 1–31.

Page, M. J., McKenzie, J. E., Bossuyt, P. M., Boutron, I., Hoffmann, T. C., Mulrow, C. D., Shamseer, L., Tetzlaff, J. M., Akl, E. A., Brennan, S. E., Chou, R., Glanville, J., Grimshaw, J. M., Hróbjartsson, A., Lalu, M. M., Li, T., Loder, E. W., Mayo-Wilson, E., McDonald, S., McGuinness, L. A., Stewart, L. A., Thomas, J., Tricco, A. C., Welch, V. A., Whiting, P., and Moher, D. (2021). The prisma 2020 statement: an updated guideline for reporting systematic reviews. *BMJ*, 372.

Purcell, W., Klipic, A., and Neubauer, T. (2022). A digital twin for grassland management. In *2022 International Conference on Electrical, Computer and Energy Technologies (ICECET)*, pages 1–6. IEEE.

Qamsane, Y., Moyne, J., Toothman, M., Kovalenko, I., Balta, E. C., Faris, J., Tilbury, D. M., and Barton, K. (2021). A methodology to develop and implement digital twin solutions for manufacturing systems. *Ieee Access*, 9:44247–44265.

Qi, Q., Tao, F., Hu, T., Anwer, N., Liu, A., Wei, Y., Wang, L., and Nee, A. (2021). Enabling technologies and tools for digital twin. *Journal of Manufacturing Systems*, 58:3–21. Digital Twin towards Smart Manufacturing and Industry 4.0.

Roque Rolo, G., Dionisio Rocha, A., Tripa, J., and Barata, J. (2021). Application of a simulation-based digital twin for predicting distributed manufacturing control system performance. *Applied Sciences*, 11(5):2202.

Semeraro, C., Lezoche, M., Panetto, H., and Dassisti, M. (2021). Digital twin paradigm: A systematic literature review. *Computers in Industry*, 130:103469.

Shafto, M., Conroy, M., Doyle, R., Glaessgen, E., Kemp, C., LeMoigne, J., and Wang, L. (2010). Draft modeling, simulation, information technology & processing roadmap. *Technology area*, 11:1–32.

Shang, D., Lang, K., and Vragov, R. (2022). A market-based approach to facilitate the organizational adoption of software component reuse strategies. *Communications of the Association for Information Systems*, 51(1):36.

Shylesh, S. (2017). A study of software development life cycle process models. In *National Conference on Reinventing Opportunities in Management, IT, and Social Sciences*, pages 534–541.

Steindl, G. and Kastner, W. (2021). Semantic microservice framework for digital twins. *Applied Sciences*, 11(12):5633.

Steindl, G., Stagl, M., Kasper, L., Kastner, W., and Hofmann, R. (2020). Generic digital twin architecture for industrial energy systems. *Applied Sciences*, 10(24):8903.

Tricco, A. C., Antony, J., Zarin, W., Strifler, L., Ghassemi, M., Ivory, J., Perrier, L., Hutton, B., Moher, D., and Straus, S. E. (2015). A scoping review of rapid review methods. *BMC medicine*, 13:1–15.

Van Dinter, R., Tekinerdogan, B., and Catal, C. (2023). Reference architecture for digital twin-based predictive maintenance systems. *Computers & Industrial Engineering*, 177:109099.

Wilking, F., Fett, M., Goetz, S., Kirchner, E., and Wartzack, S. (2024). Rsm-dt: Reusable system models for digital twins by utilizing and automating sysml profiles. In *2024 IEEE International Symposium on Systems Engineering (ISSE)*, pages 1–8. IEEE.

Xu, H., Duo, Y., and Tang, T. (2022). A digital twin framework for construction and operation of the radioactive waste repository. *Nuclear Technology and Radiation Protection*, 37(3):181–192.

Zhang, Y., Zhang, C., Yan, J., Yang, C., and Liu, Z. (2022). Rapid construction method of equipment model for discrete manufacturing digital twin workshop system. *Robotics and Computer-Integrated Manufacturing*, 75:102309.

Zhao, B., Li, X., Sun, W., Qian, J., Liu, J., Gao, M., Guan, X., Ma, Z., and Li, J. (2023). Biodt: An integrated digital-twin-based framework for intelligent biomanufacturing. *Processes*, 11(4):1213.

Zhu, Z., Xi, X., Xu, X., and Cai, Y. (2021). Digital twin-driven machining process for thin-walled part manufacturing. *Journal of Manufacturing Systems*, 59:453–466.