Evolution at the Core of Digital Twin Engineering

Tarek Alskaif[*], Önder Babur[*,‡], Francis Bordeleau[‖], Loek Cleophas[‡,¶], Benoit Combemale[**], Joachim Denil[§], Øystein Haugen[x], Judith Michael[†], Phu Nguyen[††], Tiberiu Seceleanu[‡‡], Mark van den Brand[‡], Hans Vangheluwe[§]

[*] *Wageningen University*, Wageningen, The Netherlands, Email: {tarek.alskaif, onder.babur}@wur.nl

[‡] *Eindhoven University of Technology*, Eindhoven, The Netherlands, Email: {l.g.w.a.cleophas, m.g.j.v.d.brand}@tue.nl

[‖] *École de technologie supérieure (ÉTS)*, Montréal, Canada, Email: francis.bordeleau@etsmtl.com

[¶] *Stellenbosch University*, Matieland, Republic of South Africa, Email: loek@sun.ac.za

[**] *Inria & Univ. Rennes*, Rennes, France, Email: benoit.combemale@inria.fr

[§] *Flanders Make@UAntwerpen & University of Antwerp*, Antwerp, Belgium, Email: joachim.denil@uantwerpen.be

[x] *Østfold University College*, Halden, Norway, Email: oystein.haugen@hiof.no

[†] *University of Regensburg*, Regensburg, Germany, Email: judith.michael@ur.de

[††] *SINTEF*, Oslo, Norway, Email: phu.nguyen@sintef.no

[‡‡] *Mälardalen University*, Västerås, Sweden, Email: tiberiu.seceleanu@mdu.se

***Abstract*—Engineering Digital Twins (EDT) presents a multifaceted challenge that extends beyond managing the lifecycle of a Digital Twin (DT) to include its continuous, dynamic interaction with the lifecycle of the actual object, system, or process it represents, referred to as the Actual Twin (AT). The relationship between the lifecycles of DT and AT necessitates a rethinking of the software development lifecycle of DTs. This vision paper examines the deeply intertwined lifecycles of DT and AT, arguing that effective methods for EDT must embrace the mutual and adaptive evolution of both over time. We propose placing evolution at the core of EDT. We identify key triggers of DT evolution, examine the engineering dimensions involved, and explore how the best practices, technologies, and tools of DevOps can support this evolution. Finally, we discuss current challenges and opportunities in the field. This paper serves as a call to action for the EDT community to adopt evolution as a crucial factor and core principle in EDT.**

***Index Terms*—Digital Twin Engineering, Evolution, Lifecycle, DevOps**

I. Introduction

Whatever we build today will evolve over time to accommodate changes in use, technology, regulations, etc. This continuous transformation is referred to as evolution [1].

Digital twin (DT) [2]–[4] emerged over the last decade as a key technology at the core of the digital transformation to monitor, diagnose, analyze, and optimize various aspects of systems, processes, and objects in a broad range of application domains, including aerospace, agriculture, energy, automotive, construction, healthcare, high-tech, smart cities, and telecom. In essence, a DT is a digital representation of an actual object, system, or process, referred to as the Actual Twin (AT), that is dynamically and continually (as necessary for its purposes) updated with system data, and provides a set of services related to the AT and its environment.

In spite of their growing popularity, the design and engineering of DTs often remains insufficiently prepared for evolution [5]. Most development efforts focus on short-term functionality rather than long-term adaptability. This often results in the risk that when the AT changes, the DT may need to be entirely redeveloped in order not to become obsolete [6]. This challenge reflects a deeper issue: the lack of a more global approach/methodology to support the engineering and evolution of DTs [7].

EDT is inherently a multi- and interdisciplinary process. The value provided by DTs comes not only from software engineers, but also from collaboration with domain experts from diverse backgrounds, including those without software engineering expertise. A critical aspect of this collaboration is acknowledging changes in the AT and its environment by both software engineers and domain experts. While software engineers are fully aware of software evolution, it remains unclear to what extent domain experts share this awareness [8]. This requires an interdisciplinary approach together with a co-exploration attitude.

Whether driven by new requirements, technological progress, shifting environment, user expectations, or new stakeholders, systems rarely remain static for long. For example, the shift from uni-directional to bi-directional flow of power and data in the electricity grid is an example of a major system evolution. For instance, the role of stakeholders has changed; consumers have become prosumers, and various sectors have been electrified (e.g., mobility). The rapid growth in the number of prosumers and distributed energy systems, along with the unexpectedly quickly rising adoption rates of electric vehicles, has significantly exceeded initial projections. This has triggered substantial changes in the electric infrastructure, leading to challenges such as grid congestion. The control system developed to manage the grid also needs to be adapted to accommodate the changes. Similarly, the simulators and DTs used to model, analyze and optimize the grid must be updated or redeveloped to account for new dynamics [9].

This paper, a principal outcome of the 1st Workshop on Next-Gen DT Engineering Frameworks held in May 2025 at Kasteel Wolfrath in Born (Netherlands), presents a software

engineering perspective on the evolution of DT engineering. It examines the deeply intertwined lifecycles of DTs and the AT they represent, arguing that effective methods for EDT must embrace the mutual and adaptive evolution of both over time. We propose placing evolution at the core of EDT.

The rest of the paper is organized as follows. Section II provides a background on DT lifecycles and the classes of evolution. Section III outlines the triggers of evolution. The engineering dimensions of evolution are discussed in Section IV. Section V provides a DevOps-oriented view on the evolution of DTs. Section VI identifies research challenges and opportunities related to key aspects of DT evolution and the paper is concluded in Section VII.

II. Background

For discussing the evolution of DTs, we discuss relevant background on DT architectures, the lifecycle of DTs, and different classes of evolution.

a) Digital Twin: There is a vast amount of literature that defines DT and describes DT architectures. Here, we use a high-level definition that starts with the top-level concept of a DT system (DTSys) that is composed of an AT and its environment, one or more DTs associated with the AT, and the communication between them. A DT is created for a particular purpose and provides a set of services related to its purpose. It is composed of data and a set of models used to provide the services. It is important to mention that while an AT can be composed of a mix of different types of components (including software, hardware, process, physical object, and humans), a DT is a pure software entity. Input from domain experts is crucial to build DTs that matter and are useful.

b) Digital Twin lifecycle: The DT as a software system is strongly connected to the lifecycle of the AT [10]. However, a DT has its own lifecycle as any other software system, which has to evolve over time. To support this continuous process, we rely on model-driven engineering methods for DT engineering.

c) Classes of evolution: Software maintenance and change management have been formalised through various classifications, notably, ISO 14764:2006 [11] and Kersten's software flow model [12]. ISO 14764:2006 defines software maintenance as the modification of a software product after delivery, categorizing changes into four types: *Corrective*, fixing defects; *Adaptive*, adjusting to new environments; *Perfective*, improving performance or maintainability; and *Preventive*, detecting and correcting latent faults. Complementing this technical classification, Kersten's flow model provides a value perspective by categorizing software changes into four main flow items: *Feature*, new functionality added to the product; *Defect*, fix made to correct an existing issue; *Risk*, work done to address aspects like security, privacy, and compliance exposures; and *Debt*, improvement to the software architecture and operational environment. We refer to these classes in the next section to see how our triggers fit in these classifications of changes and the value they provide.

III. Triggers of evolution

In Table I we present our classification of triggers for DT system evolution. These triggers are neither exhaustive nor disjunct, but provide a starting point to engineer a systematic framework for evolution management. Triggers may form chains: a change of purpose may cause an AT modification that requires DT updates. For example, a Combined System Purpose Change (1.2) may trigger Technical Expectations on an AT (3.2), subsequently requiring DT platform updates (4.1).

Our framework has a "Trigger Origin" dimension, identifying who initiates the evolution: *Internal–Management*, project/operational managers who recognize an issue through monitoring and oversight; *Internal–Technical*, technical team members who discover an issue during development/technical activities; *Internal–Domain*, domain experts who identify problems through specialized knowledge of the specific domain where the system works in; *External–Environment*, external forces, typically requiring a reactive response; and *External–Stakeholder*, user needs, customer requests, business stakeholder requirements.

We adapt ISO 14764:2006's maintenance categories to "evolution types" for applicability beyond software-only: *Corrective*: Responding to problems, failures, or defects after they occur. *Preventive*: Anticipating and preventing issues before they manifest. *Adaptive*: Adjusting to the environment and contextual changes. *Perfective*: Improving functionality, performance, or system qualities. We also include Kersten's flow items to capture organizational and value drivers.

Our classification of DT system evolution triggers has four main categories. The category *1. Purpose*, relates to changes in DT system goals, or not fulfilling these goals. 1.1 is about not fulfilling the purpose of the DT, defects, inaccurate models or not fully covering the requested operation. 1.2 captures the evolution of the requirements for the system from a user perspective. Some work has been done to model requirements changes and how to deal with them [13]. 1.3 looks beyond a single twin and focuses on fleets of systems, many instances of a single type, that can evolve independently [14] or resulting from specific variants of a system.

2. *DTSys* relates to changes to the whole system. 2.1 *Technical Environment Evolution* could be the deprecation of technologies, standards that evolve and need to be adhered to, or trust relations that change. 2.2 *System operation evolution*, looks at changes in the operational domain e.g. because of physical environment changes such as wear and tear, or that operational or business processes evolve over time. 2.3 *Optimisation* looks at optimisation criteria such as cost and performance.

3. *Actual Twin* looks at evolutions of the AT. 3.1 is about run-time changes. As ATs operate in the real world, both discontinuous changes (e.g., component replacement because of a failure) and continuous change (e.g., wear and tear) happen [15], [16]. 3.2 looks at the technical expectations needed of the AT. This could be triggered by 1.2, but also independently,

ID	Evolution Trigger	Trigger Origin	Evolution Type	Flow Item
1. Purpose				
1.1	**Not fulfilling purpose**			
1.1.1a	Bugs (operational monitoring)	Internal-Management	Corrective	Defect
1.1.1b	Bugs (technical discovery)	Internal-Technical	Corrective	Defect
1.1.2	Assumptions not correct	Internal-Domain	Corrective	Defect
1.1.3	Performance drift	Internal-Management	Corrective	Defect
1.2	**Combined system purpose change**	External-Stakeholder	Adaptive	Feature
1.3	**Expansion**			
1.3.1	Fleet quality drift	Internal-Management	Corrective	Risk
1.3.2	Variant-specific evolution	Internal-Management	Adaptive	Feature
2. DTSys				
2.1	**Technical environment evolution**			
2.1.1	Technological deprecation	External-Environment	Preventive	Debt
2.1.2	Standard evolution	External-Environment	Adaptive	Debt
2.1.3	Regulation evolution	External-Environment	Adaptive	Risk
2.1.4	Trust relation evolution	External-Stakeholder	Adaptive	Debt
2.2	**System operation evolution**			
2.2.1	Physical environment changes (layout, infrastructure)	External-Environment	Adaptive	Feature
2.2.2	Operational procedures change	Internal-Management	Adaptive	Feature
2.2.3	Stakeholder/user community changes	External-Stakeholder	Adaptive	Feature
2.2.4	Business process evolution	Internal-Management	Adaptive	Feature
2.3	**Optimisation (optimisation criteria in the system goal)**			
2.3.1	Reconceptualisation (from soil-based farm to hydroponic)	Internal-Technical	Perfective	Feature
2.3.2	Better precision/accuracy (part, model, sensors, data-sources, etc.)	Internal-Domain	Perfective	Feature
2.3.3	Technological improvement	Internal-Technical	Perfective	Feature
2.3.4	Secondary goals (e.g., performance improvements)	Internal-Domain	Perfective	Feature
3. Actual Twin				
3.1	**Runtime change**			
3.1.1	Discontinuous change	External-Environment	Corrective	Defect
3.1.2	Continuous change	External-Environment	Adaptive	Risk
3.2	Technical Expectations on AT	Internal-Domain	Preventive	Risk
4. Digital Twin				
4.1	**Platform Evolutions (e.g., updates)**	External-Environment	Adaptive	Risk
4.2	**Technical Expectations on DT**	Internal-Technical	Preventive	Risk
4.3	**Technical Debt Reengineering**	Internal-Technical	Perfective	Debt

TABLE I: Classification of Evolution Triggers

for example because of learning in the operation of the system (e.g., wear on bearings is a lot higher than expected).

4. *Digital Twin*, looks at triggers of the DT. 4.1 stems from the platform that needs to be monitored and managed. 4.2 looks at the technical expectations of the DT. Again, this can be triggered by e.g., 1.2, but also independently, for example when it is noticed that data volumes are higher than expected.

IV. Evolution Engineering Dimensions

DTs are software entities, so creating Digital Twin (DT) variants can be viewed similarly to creating software product lines, yet their variability is related to a broader range of aspects that are affected when evolving DTs. The dimensions we have to take into account are, e.g., the physical entity (or AT), models, data and services as introduced by Tao et al. [4], the connection between the AT and the DT as described by Kritzinger et al. [3], and different lifecycle phases of both the AT and the DT as a software system [10]. In addition, different markets have different DT variants, e.g., following national regulations, or considering various social, environmental, or ethical aspects–all coming with their own need for evolution. In the following, we focus on the engineering dimensions of evolution from a software engineering perspective (see Fig. 1).

For what purpose are we creating DTs: DT can have a wide range of different purposes, typical examples being, e.g., monitoring, behavior optimization, predictive maintenance, or validation [17]. Especially as an emergent innovative technology, the purpose of the DT can evolve as well along with the diffusion and maturation of the innovation [18], [19]. An example could be an early adopter aiming for simple data monitoring and prediction to begin with, and expanding into more advanced features such as optimization and control, this is thus related to the triggers 1.2, 1.3 and 2.2, 2.3 of Table I.

How do the involved domain scope and stakeholders evolve: While having the same purpose and the same AT, the considered scope of the DT could evolve. This includes, e.g., in a smart city DT, considering additional stakeholders such as society, as well as rightsholders [20], such as nature and future generations, which are affected by the DT. This dimension is tied with triggers under 2.2 in Table I. Further considerations on the social, legal, ethical and environmental aspects should also be taken into account, as, for instance, legislative changes could limit or disallow the use of certain data sources (e.g. as in GDPR) and models (e.g. as in AI Act) in the system.

What services must our DT support: Following essentially *all* of the evolution triggers, the services a DT is providing could evolve. This could be due to the desire to improve the quality of a service, e.g. correctness, performance etc. (triggers 1.1, 2.3, 4.3), introducing new services to support the whole DT-AT system along with its purpose, scope and environment (triggers 1.2, 2) as well as individually the AT level (e.g. new physical components) and the DT level (e.g. new platforms, infrastructure, updates) changes (triggers 3, 4). This is particularly evident in complex DT systems e.g. smart cities, where the system comprises inherently complex, dynamic and growing sub-systems/components across many domains [21].

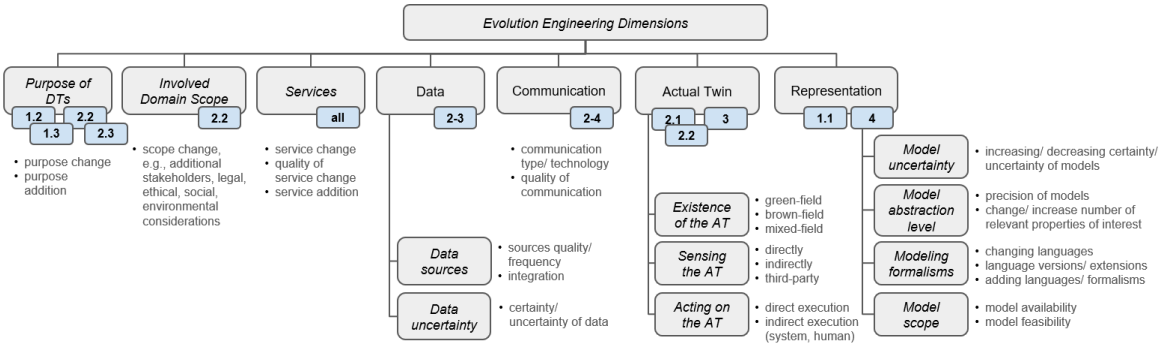

Fig. 1: Evolution Engineering Dimensions. The blue boxes show how the triggers (Table 1) influence these evolution dimensions.

Considerations on the data. (See triggers 2 and 3)

How certain or uncertain is our data? In many scenarios, data can be uncertain due to noise, incompleteness and inconsistency in the collection process, but also inherently so due to the nature of the domain [22] (see triggers 2 and 3). It should be considered how this uncertainty is measured, reported, taken into account in the rest of the DT engineering, and mitigated to some extent.

What sources of data are available? Different sources of data can become available through the evolution of the DT: sensing data on the AT (which is addressed in a separate header below), but also external data, e.g. a new training set for machine learning models used in the DT. The frequency or quality of the data can also change; e.g. moving to higher resolution satellite imagery in a smart agriculture DT setting [23]. Furthermore, in the potential scenario in a complex DT with many data sources, the process of data integration and fusion can evolve over time, e.g. with more precise and robust association and inference techniques [24].

How does communication take place? Evolution engineering of DTs should consider the type and quality of the communication in the whole system (broadly speaking triggers 2, 3 and 4). This could involve different parts (e.g., sensors/actuators, software, humans) over different communication technologies such as IoT and cloud. The technological infrastructure has implications on the rest of the system, e.g. moving from 4G to low-latency high-bandwidth 5G/6G could enable new and/or higher quality data sources, better real-time simulation and control, and so on. Finally, additional considerations could include different synchronization and security capabilities such as blockchain [25].

Considerations on the AT. (See triggers 2.1, 2.2 and 3)

Does the AT exist (yet)? Here, we can differentiate between green-field, brown-field, and mixed approaches. For *green-field approaches*, it could be helpful to consider DT requirements, e.g., for the needed data, already within the system design of the AT. For instance, one adds smart textiles in the concrete of dams to get sensory information about its inner structure [26]. For *brown-field approaches*, we have to analyze if we already obtain the needed data from the AT, or if we have to add sensors on the AT or in its environment. *Mixed-field approaches* occur, e.g., in the construction phase of a system, where the AT is gradually growing and evolving. To realize a DT for such cases, we have to ensure modularity of our DT to replace purely simulated or virtual-virtual representations with information coming from the real system (cf. software- and hardware-in-the-loop tests, where the hardware is gradually considered in the tests). Another case for mixed-field approaches is reusing parts of existing systems together with the parts describing them and providing historical information about them in a former DT, e.g., when reusing a conveyor belt of a production line within another one.

How can we sense the AT? We could sense the AT directly, e.g., via sensors, or indirectly via the environment, third-party systems. For instance, the DT of a bridge could get information from the sensors installed on the bridge [27], or alternatively receive information about its condition from the intelligent vehicles using it [28]. The type of sensing can also change over time, e.g. from numerical sensing to visual, or uni-modal to multi-modal.

How can we act on the AT? We could execute acting commands directly on the AT using actuators/controllers, or indirectly via another system, e.g., embedded programmable logic controllers on a manufacturing unit, another DT, or a human, as the human-in-the-loop.

System representation. (See triggers 1.1 and 4).

What can we model? At any point of the DT evolution, a (re-)assessment of the availability and feasibility of existing models to be used might be required. Even if models remain the same, the underlying assumptions and validity frames [29] of the models can change, in parallel to the data evolution (e.g., via collecting new data and insights as a result), leading to changes in the models application and precision. Similarly, over the life span of the DT even different modelling paradigms might be employed, e.g., from simulation models in early phases where data is scarce, to more data-driven and hybrid approaches in further iterations.

Which modeling formalisms are used in the domain under study? Another consideration is whether to move from using ad-hoc models (e.g., in Python) into more formal, domain-specific models. Even sticking to specific formalisms, the modeling languages themselves might evolve over time [30].

Which abstraction level do we need for modeling? Evolution might require changing the abstraction level of existing

models, e.g., to be more precise in simulation models for cars based on a change of legislation for driver support.

How certain or uncertain are our models? Based on how uncertain the models we have about an actual system are, our developed DT requires mitigation strategies to cope with this uncertainty and with evolving uncertainties, which could be both increasing and decreasing, both in terms of computational uncertainty [31] and belief uncertainty [32].

In summary, there are many dimensions to consider in engineering evolving DTs, linked to the various triggers in the previous section. These dimensions are not per se orthogonal, but rather interrelated and interdependent. A successful engineering process can be achieved by accounting for these dimensions throughout the lifecycle of the DT, from exploration and early design to iterative adaptation and optimization.

V. DevOps and evolution

To enable systematic evolution, EDT must leverage the best practices of software engineering [33], Agile and DevOps ([34], [35]), including: best practices of software architecture to support evolution; integration of the software engineering lifecycle phases in a unified iterative process focused on continuous improvement and evolution; and use of techniques and tools to automate as many software process tasks as possible to enable fast and reliable delivery of software updates. The goal is to make EDT as agile, systematic, and reliable as possible.

Fig. 2: DevOps lifecycle loop

Architecture. A modular architecture is essential to support the engineering and evolution of DTs. In particular, the architecture must support key principles such as: separation of concerns, allowing teams to work independently on specific components; locality of change, minimizing the impact of modifications; and support for the independent development and deployment of small, self-contained code units, enabling easier testing and rapid delivery of software updates.

To facilitate evolution, DevOps promotes the use of component-based [36] and microservices architectures [37]. Additionally, organizing components or microservices, related to different DT aspects, into layers or larger aggregates enables independent management of functionalities such as AT data acquisition, AT control, data storage, internal DT communication, DT services, and visualization. This approach also facilitates the grouping of services related to specific DT aspects or expertise domains into service clusters, enhancing modularity and maintainability.

DevOps process. To support evolution, EDT should adopt an iterative DevOps approach focused on continual learning and experimentation through automation and monitoring at all stages of the software lifecycle (see Fig. 2), which is composed of four main phases: Plan, Develop, Deliver, and Operate. Each lifecycle iteration must be viewed as an experiment and an opportunity to learn.

Systematic evolution requires careful planning and management of each iteration. Iteration content is drawn from a prioritized backlog of tasks, created in response to evolution triggers (Table I) and tracked on a product management board (e.g. Kanban). Depending on priority, tasks may be handled immediately (e.g. urgent bug fix) or deferred to future iterations.

In addition to the conventional activities included in each phase, we identify key activities that must be added in each phase to support systematic DT evolution.

Plan: The goal of this phase is to define the content of the next iteration as a set of prioritized tasks. However, before doing so, the state of the DT system must be assessed using data and metrics from the Operate phase. This analysis allows identifying issues and improvement opportunities, leading to the creation of new tasks to be added to the product backlog.

The content of each iteration is defined based on two main constraints: available development resources, and percentage of resources allocated to each of the four flow items (Feature, Defect, Risk, and Debt) identified in Table I. It is essential to maintain a proper balance among the four flow items over time to ensure sustainable progress and system health.

To ensure systematic evolution, each development task should have clearly defined, measurable objectives. This enables the implementation of telemetry in the Develop phase and the tracking of its impact during the Operate phase.

Develop: The goal of this phase is to implement the tasks defined for the current cycle to produce a new version of the DT. During this phase, new versions of the components (microservices) are developed and thoroughly tested, including integration, product, and system-level testing, across various testing and pre-production environments to ensure quality. Also, deployment requirements must be addressed to ensure the efficient and reliable deployment and operation of the DT.

To support systematic evolution and align with DevOps best practices, product telemetry must also be implemented to enable effective monitoring during the Operate phase. This ensures that the new DT version is achieving the objectives defined in the Plan phase. For this purpose, we need to collect the relevant data and implement the necessary models and algorithms to compute the required telemetry metrics.

Deliver: The goal of this phase is to release a new version of the DT software. To support systematic evolution, deployment to the production environment should be automated as much as possible through DevOps pipelines. While automated deployment in cloud environments is now standard, deploying to fog and edge nodes remains more challenging due to security and performance constraints. In situations where full automation is not feasible, the deployment process should still be systematic and well-documented to ensure repeatability and consistency.

Operate: The main goal of this phase is to manage DT operations and monitor its performance through telemetry, ensuring proper functioning and enabling the identification of issues and improvement opportunities. During this phase, execution data are collected and metrics are computed to monitor the DT evolution against the improvement objectives. The Monitor phase loops back into the Plan phase, during which the telemetry results are analyzed.

Tools and Automation. Frequent deployment accelerates issue detection and learning, making software process automation a key driver of DT evolution. To improve both the quality and frequency of DT software deployments, DevOps technologies and tools should be leveraged to automate as many software process tasks as possible. This includes: Git, CI/CD pipeline technologies, automated testing tools, container technologies, and deployment management technologies. It also includes different technologies and tools to support the creation and analysis of telemetry, e.g. OpenTelemetry [38], Prometheus [39], Grafana [40], and ELK [41]. These technologies enhance flexibility and scalability, enabling efficient deployment and maintenance of DT components. They also allow domain experts to apply their knowledge without needing software engineering expertise, by leveraging DevOps best practices and avoiding the complexities of integration, testing, and deployment, ultimately accelerating DT evolution.

To manage the evolution of data models/ML workflows, techniques developed for DataOps/MLOps/ModelOps can be used within the DT, e.g., CI/CD workflows, software containers, and model registries to streamline AI-driven operations.

Finally, rollback mechanisms should be implemented to automatically revert to a stable version of the DT software when new modifications introduce operational issues. In such cases, telemetry data collected during the operations of the faulty version must be analyzed to identify the root cause. Additionally, deployment strategies such as blue-green deployment, canary releases, and A/B testing can be used to better control and evaluate the evolution of the DT.

VI. Challenges and Opportunities

In addition to the different points discussed in the previous sections, we identify the following research challenges and opportunities related to key aspects of DT evolution.

Partial knowledge. As any software engineering process, EDT starts with partial knowledge. This is a fundamental reason why systematic evolution needs to be considered from the beginning. In addition to the conventional partial knowledge related to the user and system needs and requirements, EDT has to deal with many other sources of partial knowledge, including the ones related to the evolution of the AT, and ones related to the available data and their quality that have a direct impact on the DT services. To address this challenge, techniques like the ones developed to manage uncertainty ([22], [31], [32], [42]) should be integrated in the EDT process.

Co-evolution of AT and DT. The co-evolution of the DT and AT as parts of an integrated DT system is a key aspect of DT evolution, yet often overlooked in existing methodologies. Since AT and DT evolution are typically managed by different people/roles, clear responsibilities must be defined. As discussed in Sections III and IV, changes in the AT can trigger DT evolution and vice versa. Further research is needed to make this co-evolution more systematic.

DT architecture. DT architectures have been developed across various purposes, contexts, and domains in recent years. These architectures need to be evaluated for their ability to support evolution, particularly in maintaining model and data synchrony with the evolving AT and meeting changing DT requirements, such as service precision. Additionally, they should support DT composition to enable the iterative and incremental evolution of the DT's purpose [13].

DT Development methodology. Section V discusses the use of DevOps to support DT engineering and evolution. However, beyond current state-of-the-art and state-of-practice, more research is required to provide first-level support for DT evolution telemetry and multidisciplinary team collaboration.

Resource consumption. In the context of sustainable development, it is essential to consider the overall resource consumption of the DT system. Specifically, the DT's resource use should be minimized and must not outweigh the improvements it enables in the AT. For example, if a DT is designed to reduce the energy consumption of an AT, such as a building, its own energy use should not exceed the savings it generates.

Open source initiative. In recent years, various open-source technologies have emerged to support different aspects of DT engineering. A review of these projects is needed to assess the current state-of-the-art and state-of-practice in supporting DT evolution. This should be followed by a reflection on how the research and engineering communities can collaborate to advance DT evolution, either by contributing to existing projects or initiating new ones.

VII. Conclusion

We focused on identifying and classifying the sources of DT evolution. Evolution of DTs cannot occur in isolation but must be closely aligned with their corresponding ATs and operating environments. As DTs span the entire system lifecycle, evolution should be central to EDT and a shared concern across disciplines, including for domain experts, developers, and operators. Building on existing research and new insights, we identify key challenges and opportunities.

We hope this contribution will inspire future cross-disciplinary efforts to promote and implement evolvable DT systems. Further exploration at lower abstraction levels, across areas such as technology, safety, security, and communication, can foster bottom-up, cross-domain awareness analysis and specification.

Acknowledgments

This research has been partly funded by the EU and KDT Joint Undertaking through MATISSE Project GA n. 101140216, the Natural Sciences and Engineering Research Council of Canada (NSERC), and the DFG and ANR project – Model-Based DevOps – 505496753.

References

[1] M. Pennotti, P. Brook, and D. Rousseau, "The evolution of systems engineering as a transdiscipline," *Systems Engineering*, vol. 27, pp. 899–910, 04 2024.

[2] M. Grieves and J. Vickers, *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*. Springer International Publishing, 2017, pp. 85–113.

[3] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital Twin in manufacturing: A categorical literature review and classification," *Ifac-PapersOnline*, vol. 51, no. 11, pp. 1016–1022, 2018.

[4] F. Tao, W. Liu, M. Zhang, T.-l. Hu, Q. Qi, H. Zhang, F. Sui, T. Wang, H. Xu, Z. Huang *et al.*, "Five-dimension digital twin model and its ten applications," *Computer integrated manufacturing systems*, vol. 25, no. 1, pp. 1–18, 2019.

[5] J. Michael, I. David, and D. Bork, "Digital Twin Evolution for Sustainable Smart Ecosystems," in *Workshop on Models and Evolution, MODELS Companion '24: International Conference on Model Driven Engineering Languages and Systems (ME)*. ACM, 2024, pp. 1061–1065.

[6] F. Bordeleau, B. Combemale, R. Eramo, M. Van Den Brand, and M. Wimmer, "Towards model-driven digital twin engineering: Current opportunities and future challenges," in *International conference on systems modelling and management*. Springer, 2020, pp. 43–54.

[7] H. M. Muctadir, D. A. M. Negrin, R. Gunasekaran, L. Cleophas, M. van den Brand, and B. R. Haverkort, "Current trends in digital twin development, maintenance, and operation: an interview study," *Softw. Syst. Model.*, vol. 23, no. 5, pp. 1275–1305, 2024. [Online]. Available: https://doi.org/10.1007/s10270-024-01167-z

[8] S. Terry and V. Chandrasekar, "Systems engineering barriers to legacy system evolution: Legacy system assessment," *Systems Engineering*, vol. 28, pp. 207–223, 10 2024.

[9] W. Zomerdijk, P. Palensky, T. AlSkaif, and P. P. Vergara, "On future power system digital twins: A vision towards a standard architecture," *Digital Twins and Applications*, vol. 1, no. 2, pp. 103–117, 2024.

[10] J. Michael, L. Cleophas, S. Zschaler, T. Clark, B. Combemale, T. Godfrey, D. Khelladi, V. Kulkarni, D. Lehner, B. Rumpe, M. Wimmer, A. Wortmann, S. Ali, B. Barn, I. Barosan, N. Bencomo, F. Bordeleau, G. Grossmann, G. Karsai, O. Kopp, B. Mitschang, P. Muñoz Ariza, A. Pierantonio, F. Polack, M. Riebisch, H. Schlingloff, M. Stumptner, A. Vallecillo, M. van den Brand, and H. Vangheluwe, "Model-Driven Engineering for Digital Twins: Opportunities and Challenges," *INCOSE Systems Engineering*, 2025. [Online]. Available: https://doi.org/10.1002/sys.21815

[11] I. Standard, "Software engineering–software life cycle processes–maintenance," *ISO Standard*, vol. 14764, p. 2006, 2006.

[12] M. Kersten, "What flows through a software value stream?" *IEEE Software*, vol. 35, no. 4, pp. 8–11, 2018.

[13] J. Mertens, S. Klikovits, F. Bordeleau, J. Denil, and Ø. Haugen, "Continuous evolution of digital twins using the dartwin notation," *Software and Systems Modeling*, pp. 1–22, 2024.

[14] J. Mertens and J. Denil, "The digital twin as a common knowledge base in devops to support continuous system evolution," in *Computer Safety, Reliability, and Security. SAFECOMP 2021 Workshops*. Springer International Publishing, 2021, pp. 158–170.

[15] P. Muñoz, M. Wimmer, J. Troya, and A. Vallecillo, "Using trace alignments for measuring the similarity between a physical and its digital twin," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '22. ACM, 2022, p. 503–510. [Online]. Available: https://doi.org/10.1145/3550356.3563135

[16] J. Mertens and J. Denil, "Reusing model validation methods for the continuous validation of digital twins of cyber-physical systems," *Software and Systems Modeling*, pp. 1–23, 2024.

[17] 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 (JSS)*, vol. 193, November 2022.

[18] G. Orr, "Diffusion of innovations, by everett rogers (1995)," *Retrieved January*, vol. 21, p. 2005, 2003.

[19] L. Tudorache, Ö. Babur, S. S. Lucas, and M. van den Brand, "Current approaches to digital twins in additive manufacturing: a systematic literature review," *Progress in Additive Manufacturing*, pp. 1–35, 2025.

[20] K. Klomp, *Betekeniseconomie: de waarde van verweven leven*, 2022.

[21] N. Mohammadi and J. E. Taylor, "Smart city digital twins," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2017, pp. 1–5.

[22] R. H. Hariri, E. M. Fredericks, and K. M. Bowers, "Uncertainty in big data analytics: survey, opportunities, and challenges," *Journal of Big data*, vol. 6, no. 1, pp. 1–16, 2019.

[23] R. K. Goel, C. S. Yadav, S. Vishnoi, and R. Rastogi, "Smart agriculture–urgent need of the day in developing countries," *Sustainable Computing: Informatics and Systems*, vol. 30, p. 100512, 2021.

[24] F. Castanedo, "A review of data fusion techniques," *The scientific world journal*, vol. 2013, no. 1, p. 704504, 2013.

[25] T. H. Luan, R. Liu, L. Gao, R. Li, and H. Zhou, "The paradigm of digital twin communications," *arXiv preprint arXiv:2105.07182*, 2021.

[26] J. Michael, J. Blankenbach, J. Derksen, B. Finklenburg, R. Fuentes, T. Gries, S. Hendiani, S. Herlé, S. Hesseler, M. Kimm, J. C. Kirchhof, B. Rumpe, H. Schüttrumpf, and G. Walther, "Integrating Models of Civil Structures in Digital Twins: State-of-the-Art and Challenges," *Journal of Infrastructure Intelligence and Resilience*, vol. 3, no. 3, 2024.

[27] C. Ye, L. Butler, C. Bartek, M. Iangurazov, Q. Lu, A. Gregory, M. Girolami, and C. Middleton, "A digital twin of bridges for structural health monitoring," in *12th International Workshop on Structural Health Monitoring 2019*. Stanford University, 2019.

[28] Z. Hu, S. Lou, Y. Xing, X. Wang, D. Cao, and C. Lv, "Review and perspectives on driver digital twin and its enabling technologies for intelligent vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 417–440, 2022.

[29] J. Denil, S. Klikovits, P. J. Mosterman, A. Vallecillo, and H. Vangheluwe, "The experiment model and validity frame in m&s." in *SpringSim (TMS)*, 2017, pp. 10–1.

[30] B. Meyers and H. Vangheluwe, "A framework for evolution of modelling languages," *Science of Computer Programming*, vol. 76, no. 12, pp. 1223–1246, 2011.

[31] W. L. Oberkampf, S. M. DeLand, B. M. Rutherford, K. V. Diegert, and K. F. Alvin, "Error and uncertainty in modeling and simulation," *Reliability Engineering & System Safety*, vol. 75, no. 3, pp. 333–357, 2002.

[32] L. Burgueño, P. Muñoz, R. Clarisó, J. Cabot, S. Gérard, and A. Vallecillo, "Dealing with belief uncertainty in domain models," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–34, 2023.

[33] H. Washizaki, "An overview of the swebok guide." [Online]. Available: https://sebokwiki.org/wiki/An_Overview_of_the_SWEBOK_Guide

[34] G. Kim, J. Humble, P. Debois, J. Willis, and N. Forsgren, *The DevOps handbook: How to create world-class agility, reliability, & security in technology organizations*. San Francisco: IT Revolution, 2021.

[35] S. Aissat, J. Beaulieu, F. Bordeleau, J. Gascon-Samson, E. A. Poirier, and A. Motamedi, "JuNo-OPS: A DevOps framework for the engineering of digital twins for built assets," in *ACM/IEEE 27th Int. Conf. on Model Driven Engineering Languages and Systems*, 2024, pp. 496–506.

[36] G. T. Heineman and W. T. Councill, "Component-based software engineering," *Putting the pieces together, addison-westley*, vol. 5, no. 1, 2001.

[37] S. Newman, *Building microservices: designing fine-grained systems*. O'Reilly Media, Inc., 2021.

[38] D. G. Blanco, *Practical OpenTelemetry*. Springer, 2023.

[39] J. Turnbull, *Monitoring with Prometheus*. Turnbull Press, 2018.

[40] M. Chakraborty and A. P. Kundan, "Grafana," in *Monitoring cloud-native applications: Lead agile operations confidently using open source software*. Springer, 2021, pp. 187–240.

[41] S. Chhajed, *Learning ELK stack*. Packt Publishing Ltd, 2015.

[42] A. Thelen, X. Zhang, O. Fink, Y. Lu, S. Ghosh, B. D. Youn, M. D. Todd, S. Mahadevan, C. Hu, and Z. Hu, "A comprehensive review of digital twin—part 2: roles of uncertainty quantification and optimization, a battery digital twin, and perspectives," *Structural and multidisciplinary optimization*, vol. 66, no. 1, p. 1, 2023.