Model-driven Software Engineering in practice: (SPML4AI) a Framework for Modeling and Executing Software Engineering Process for AI-Enabled Systems

***Abstract*—Software systems with an AI component are currently under demand and are being investigated in several domains. The process of developing an AI-enabled (data-intensive intelligent) software differs from traditional (non AI-enabled) software systems. Traditional software engineering involves implementing an executable program which principally includes writing code. On the other hand, engineering software with an AI component involves novel processes that require substantial degree of flexibility, such as managing data and training models, with relatively less emphasis on the code writing. While solving software engineering problems with machine learning techniques (AI4SE) has been largely studied and applied, more recently there has been increasing interest in how to improve the engineering of systems with AI components (SE4AI). Work on this topic is scattered across different communities, including the modeling community. Existing process modeling language, such as SPEM are unable to fully meet the characteristics of software engineering for AI-enabled system (e.g., continuous learning and adaptation). To fill this gap, trade-offs between software process modeling languages in AI/ML context must be re-examined. In this paper, we propose a framework for modeling and executing software engineering process for AI-enabled system.**

***Index Terms*—Software Process Modeling, Traditional Software Engineering Process, Software Engineering for AI-enabled System, Artifact-Centric Processes Modeling**

I. Introduction

Lonchamp [1] defines software process as *"a set of partially ordered process steps, with sets of related artifacts, human and computerized resources, organizational structures and constraints, intended to produce the requested software deliverable"*.

Software process model can be defined as a representation or framework used to describe the activities, tasks, roles and artifacts involved in the software development life cycle. The software process model brings a set of benefits for organizations, including, providing a structured view of the processes, facilitating improvements, and permitting of the processes standardization and reuse. In order to build a software process model, it is necessary a process modeling language to represent the elements that compose a software process. Two primary approaches exist for process modeling, activity-centric and artifact-centric. The activity-centric centers around activities as the main process elements, representing processes as a sequence of activities and tasks in a coarse-grained manner. Conversely, the artifact-centric prioritizes artifacts as the primary process elements, allowing for more precise results by incorporating finer-grained elements.
In order to specify the behavior and rules of the process model, imperative and declarative methods are distinct. In imperative modeling, the focus lies in explicitly defining the sequence of elements needed for process execution, requiring any new alternatives or variations to be incorporated during the design phase. On the other hand, declarative modeling emphasizes defining the desired process outcome without providing a detailed, step-by-step procedure. The choice between the modeling approach and method depends on the nature of the processes and the level of control and flexibility required.

Nowadays, Artificial Intelligence-enabled Systems (AIS) are used increasingly in real-world applications. This is mainly due to the success of deep learning algorithms in the fields of image processing, speech recognition and machine translation. In the context of software engineering, developing software systems with AI components present new challenges. AIS is a software system including one or more ML components. Unlike traditional software systems, in which developers explicitly program the systems' behaviour, The behaviour of ML components are defined jointly by the code that implements them and the data used for training them. Due to this changes of the development paradigm, new demands changes in the development process arose. We argue that the changes that ML development provide to the development processes bring a unique set of requirements to the modeling perspective. Addressing the challenges of the development paradigm shift into process modeling, gave rise to the following research questions: *Q1: What challenges emerge in the management of software engineering processes for AI-enabled systems? Q2: What are the essential characteristics that process modeling should encompass to address the novel challenges introduced by AI-based systems development processes?*

II. Problem Statement

A. Motivation

As the machine learning community continues to accumulate years of experience with live systems, a wide-spread software engineering studies for developing machine learning applications has emerged [2]–[4]. Developing and deploying AIS is relatively fast, but maintaining them over time is

difficult and expensive [5]. An Artificial Intelligence-Enabled System (AIS) is a software system in which at least one of its components relies on one or more ML models. ML model can be defined as *"a trained instance of a specific machine learning algorithm"* [6]. ML Model is ultimately a software module [7] with special behaviours defined jointly by the algorithm that implements it and the data used for training it. There are three primary reasons why the ML module differs from traditional software module:

Data dependency: ML module is heavily rely on data. I.e., ML module's behaviors are determined by the characteristics and patterns within the data itself. Software module, on the other hand, is typically utilize data as inputs to perform specific operations. However, the functionality of the software module itself is not inherently reliant on the data.

Adaptability: ML module can be retrained with new data to improve their performance. It has the ability to learn from new data and adjust its behavior correspondingly. Traditional software module, once deployed, often requires separate updates and modifications to incorporate changes in functionality.

Interpretability: ML module often lack interpretability, making it challenging to explain their decision-making process. Traditional software module, in turn, typically has clear and understandable code that allows developers to analyze its behavior.

Nowadays, it is common to integrate one or multiple ML components within a software system. Integrating existing Software Engineering (SE) processes with ML-specific workflow have brought to development processes two main novel activities required to develop ML components, namely, Managing Data and Training ML model [2]. Due to this integration, new demands changes in Requirement Engineering (RE) [8] and Software Testing (ST) processes [6] occurred. The integration of ML-specific workflow and the SE process is presented in Fig. 1. Besides one or more machine learning models, an AIS may embraces other software modules, responsible e.g., for input transformation, user interaction or functional logic which is not inferred from the training data [6].

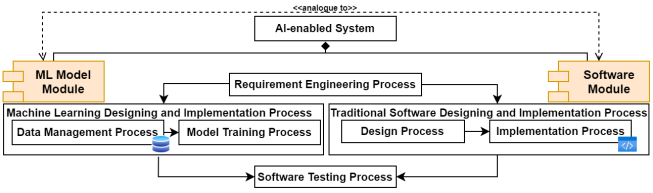

Fig. 1. The integration of ML-specific workflow and the Software Engineering processes. Analogy between ML Model Module and Software Module

Requirement Engineering (RE) is the process of eliciting, analyzing, specifying, validating the Functional Requirements (FRs) and the Non-Functional Requirements (NFRs) of the software system. In the development of AIS, it is essential not only to define the requirements for the final product but also to establish the requirements for the learning process. For instance, ML designer must decide which features are feasible to implement and decide the types of the appropriate models for the given problem [2]. The NFRs of AIS include measuring the performance, reliability, traceability, and usability but with scope and importance different than the traditional software system [3], [9]. I.e., in contrast to the traditional system, AI' NFRs can be defined and measured over different granularity levels of the system [10]. In particular over the following:

(1) The ML algorithm that performs the learning task.
(2) The training data.
(3) The ML model trained using the data.
(4) The results of applying the ML model.
(5) The entire AI system.

The unique characteristics of AIS, including model complexity and dynamic data environments, contribute to the increased difficulty of maintaining the NFRs. Hence, it can be stated that the effective satisfaction of NFRs becomes critical [11].

Software Testing (ST) is the process of verifying that a software behaves as specified, detecting errors, and validating that what has been specified is what the stakeholders actually needed [12]. Typically ST has different test levels, including unit, integration, and system level testing. In the case of developing an AIS, the same tests also applied, but with further specialisation. [6]. To better capture the specific properties of AIS testing, Riccio et al. [6] distinguish between four types of SE for AIS's testing:

(1) Input-level Testing: analyse the data used to train the ML model as well as the input data used at the prediction time.
(2) Model-level Testing: aim at finding faults due to the suboptimal model architectures, and training process.
(3) Integration Testing : test the combined software components (including ML models) and hardware components to evaluate the interaction between them.
(4) System Testing: evaluate the AI compliance with its specified. requirements.

While there can be some overlap between traditional SE and SE for AIS in certain areas, their primary purpose distinguish them from each other. Among the challenges in managing the AI development process include the maintaining of the *Traceability*. Traceability imposes the conditions that the interdependencies among the artifacts be made explicit and that each artifact be trackable longitudinally through the entire development process. This becomes especially complex with ML components, where traditional issues still exist, but now we must address data artifacts at a more granular level.

B. Running Example

In order to investigate whether current Process Modeling Languages (PMLs) offer support for effectively model SE processes for AIS, we use two standard languages, namely SPEM [13] and BPMN [14]. We provide partial model of ML workflow practiced by Microsoft team. Microsoft' ML workflow is nine stages with some data-oriented stages (e.g., data cleaning, and data labeling) and others model-oriented (e.g., feature engineering, and model training) [2]. Figure 2 illustrates the outcome of one examination experience employing SPEM. As illustrated, Data Scientist takes part in the RE [9], and collaborates with Requirements Engineer, Domain

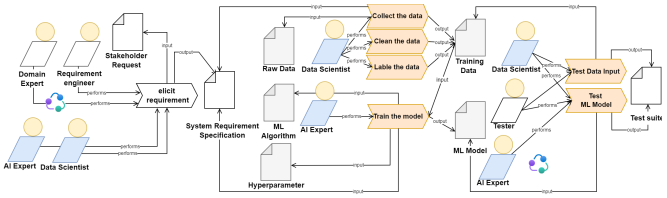

Fig. 2. SPEM v2.0 Model of partial Microsoft' ML workflow. AIS-related roles in blue color, -related tasks in Orange and -related artifacts in gray

Expert and AI Developer to specify the data requirements (e.g., data format and range). Data requirements serve as an input for the Data Scientist- during the data management process- who analyzes the given dataset based on the specified requirements. From this example, it can be inferred that the new momentum AIS provides to the process elements include:

(1) Joint effort of roles from different backgrounds involved in the development process (e.g., data scientist, ML experts). One obstacle stemming from the stability of ML models is the unreliability of predictive results in the face of minor changes in the situation. The challenges stand for urgent problems to be solved from each role [9].

(2) Artifacts of novel types are emerged (e.g., training data, ML model). ML artifacts often include data-related components, such as datasets and feature engineering pipelines. Ensuring artifacts compatibility (the same set of dependencies and versions are used consistently across different platforms and collaborators) can be challenging in AIS. This is because data scientists are quite often continue to tweak the ML model while it is being deployed [9]. Changes in dependencies or versions may introduce variations in behavior. Compatibility ensures that the ML components necessary for achieving reproducibility work together without conflicts. Additionally, proper traceability significantly increases the likelihood of achieving reproducibility, as the traceability adds transparency making it easier to identify potential errors or biases.

(3) Novel task types are emerged (e.g., clean data, train ML model). Among the challenges that arise from the ML-related tasks include the refinement of the ML model to achieve the desired performance. This process is important in order to improve the accuracy and robustness of the AIS. Refining poses a significant challenge, including the need to revisit the data management, feature engineering, and model training steps multiple times to experiment with different approaches and refine the ML solution.

C. Observations and Suggestions

In this section, we explore the key insights derived from the examination of modeling SE for AIS using SPEM and BPMN. The examination focuses on evaluating their applicability to support SE for AIS. The evaluation shed light on various aspects, including the ability of the modeling approaches to handle complex relationships between SE artifacts and AIS components, their support for iterative and incremental development, their capacity to handle data-centric aspects of AIS, and their ability to address traceability concerns.

Among the different key factors that contribute to the success of software development projects, *Traceability* and *Flexibility* are essential aspects that help ensure the efficiency, reliability, and adaptability of the software throughout its life cycle. *Traceability* ensures a clear understanding of the relationships between artifacts and the history of decisions made during software development, while *Flexibility* empowers the team to respond to changes and deliver software that remains relevant and valuable in a dynamic environment. In this study, we specifically concentrate on these two factors to assess the suitability of SPEM and BPMN to support SE for AIS.

Obs.1: The two PMLs support coarse-grained modeling: the processes are described at a larger scale or level of abstraction, such that the details of individual process elements are reduced. The choice between coarse-grained and fine-grained modeling involves a trade-off. Balancing computational efficiency gained from higher abstraction levels with the potential loss of accuracy or precision in capturing fine-scale dynamics is crucial. The decision largely depends on the application goals and the level of detail required to effectively address the problem at hand. From a *Traceability* perspective, precise details are necessary to enable smooth tracing among artifacts.

As highlighted in the previous Section, we gained insights into the possibility of applying NFRs of ML at varying levels of granularity. Take for instance the case of Training Data Artifact, which is commonly divided into three subsets, illustrated in Fig. 3: the Training Set, used to train the ML model; the Test Set, employed to evaluate the ML model's performance and its ability to recognize patterns in new data; and the Validation Set, utilized to fine-tune the model's hyperparameter. In order to establish clear trace relations between (e.g., hyperparameter and validation set), it is necessary to decompose Training Data artifact into its constituent parts.

Obs.2: The collaboration of participants with different expertise (data scientists, ML experts, etc.) is considered as a major challenge in AIS, as the bridging semantical gaps between different knowledge is important [9].

Obs.3: The two PMLs support the imperative modeling scheme for processes. In the context of AIS, the AIS development processes are highly non-linear and contain several feedback loops [2]. This iterative process allows for refinement and improvement of the ML model over multiple iterations. For instance, when an ML expert observes a significant distribution shift between the training data and real-world data, they may decide to revisit the data collection process to obtain more representative data and reiterate the ML workflow (Data Augmentation) [2]. Another example involves adjusting the model parameters or exploring alternative algorithms (Model Tuning).

Obs.4: The two PMLs do not have inherent execution capabilities. Modeling SE for AIS processes shall foster the understanding of how these processes operate. During process enactment, it should be also possible to maintain the effective and qualitative traceability relationships (so that stakeholder could be easily track the requirements from the given artifacts). Following, we propose various suggestions aimed at

supporting the aforementioned observations:
Sugg.1: regarding Obs.1, our findings suggest that adopting an Artifact-centric approach can enhance *Traceability* by establishing finer-grained trace links between the artifacts. In relation to Obs.2, the use of Artifact-centric facilitates the utilization of standardized terminology across different participants involved in the processes.
Sugg.2: due to the iterative nature of the AIS development process, it is recommended to adopt a declarative approach. This reduces the future cost of making adjustments to production decisions in response to changes in the external environment.
Sugg.3 : in order to enable a holistic support for SE for AIS, AI-specific process elements need to be defined, in order to allow modeling and executing the AI-specific aspects of SE processes.

III. Expected contributions

We present the following propositions to address the suggestions highlighted in the previous Section:
(1) Define Domain-specific modeling (DSL) for creating models that are specific to SE for AIS (including the specific related AI tasks, roles, and artifacts). Defining these concepts proves beneficial for enhancing communication among different participants, automating trace link generation, enabling more flexible execution, and facilitating partial process modeling.
(2) Define an Artifact-centric process modeling (ACPM) methodology. From our findings, the literature lacks concrete guidelines for modelers to systematically define artifact-centric process models with the adequate details to enable an effective control of process execution. We address this lack by proposing a goal-oriented ACPM methodology for systematically modeling the processes [15].
(3) Develop a data-driven process engine capable of executing fragmented artifact process models, and generating a macro model derived from the partially modeled process.

IV. Related Works

In this section, we highlight relevant related work topics, focusing on the artifact-centric and activity-centric process modeling language (encompass both imperative and declarative approaches).
First: Activity-centric Process Modeling Language:
an approach emphasis on the activities or tasks to model the process. Example of Activity-centric PMLs include the following: (A) Software Process Engineering Metamodel (SPEM) [13]: a standardized software engineering processes modeling language. In SPEM, activities are the primary elements used to model the software process. While activities are the primary focus of SPEM, it allows for the explicit modeling of artifacts and their relationships through the ”workProduct” construct. However, SPEM supports a higher-level decomposition of the artifacts. If the process is decomposed into a large number of fine-grained elements, it may introduce additional relationships and dependencies; which can potentially increase the complexity of the overall model. Therefore, the relationships need to be properly structured to avoid unintended complexity. (B) Business Process Model and Notation (BPMN) [14] a standardized business processes modeling language. BPMN's primary focus is on modeling the flow of activities and events within a business process. While BPMN does not provide a dedicated and standardized way to model artifacts, the use of data objects and annotations can help in representing artifacts indirectly within the context of a BPMN diagram.
Second: Artifact-centric process modeling (ACPM):
an emerging approach introduced by the Business Process Management community. ACPM is a data-centric approach aims to have an integrated perspective on process by coupling artifacts and activities to specify the process [16]. The common ground for ACPM can be described by the BALSA framework, we present a brief description of its architecture.
BALSA framework consists of: Business Artifact(represents a key business-relevant object used in the process. Each artifact is characterized by data attributes; Life-cycle (expresses the evolution of an artifact, from its creation until its termination); Task (an action manipulating an artifact and making it evolve); and Association(associates a Task with an Artifact to represent the impact of the task on the evolution of the artifact. The association can be sequential or constraints-based).
Example of ACPM include the following languages:
(A) PHILharmonic Flows [17]: is a framework aims to address the modeling and execution of distributed processes based on BALSA framework. Another framework proposed by Kun et al. [18] for BALSA framework with a bottom-up abstraction mechanism. Both frameworks provide an Imperative BALSA-based ACPML. (B) Guard-Stage-Milestone (GSM) [19], is a declarative BALSA-based ACPM language, which composed of artifact's life-cycle evolved based on Event-Condition-Actions rules. (C) Case Management and Notation (CMMN): non BALSA-based ACPML, emphasises on the modeling of artifacts (data, documents, objects, etc.) as a flow of cases, where a ”case” represents a specific instance of work or a unique situation [20]. We refer to [21] as another examples of non BALSA-based ACPML.

In practice, SE processes are often modeled using a combination of general-purpose process modeling languages (such as SPEM™ or UML®) and adapted methodologies that incorporate artifact-centric principles. However, there is no exclusive artifact-centric process modeling langugae dedicated to SE processes.

V. Proposed Approach

In response to the arguments provided in the Observations and Suggestions Section, we propose *SPML4AI*, a framework to model and execute SE for AIS, with the following main characteristics: Artifact-Centric Paradigm and Constraints-based Scheme. Reviewing the literature, we found that there are different approaches that deal with the modeling processes based on artifact-centric in the business community [16], [17], [21]. However, the structure of business and software artifacts differs significantly due to their distinct natures and purposes(e.g., business artifact's structure may be defined and

fixed through templates or standards, while software artifact' structure is more dynamic and may evolve over the software development life cycle). Let's consider design refactoring as an example, refactoring process requires restructuring the code to improve its maintainability. Additionally, due to the iterative nature of software development in general, and the AIS in particular, the incremental changes may result in modifications to the structure of existing artifacts.

We adopt GSM [19], a constraint-based artifact-centric business process modeling language to model SE for AIS. As can be observed in the application scenario Fig. 4, GSM provides detailed micro-level process models, we found out that it is necessary to complement the micro models with a macro-level representations. Therefore, our approach aims to support ACPM at two levels: Macro Process Model (representing artifact relationships) Fig. 3 and Micro Process Model (for individual artifacts) Fig. 4. In addition, due to the absence of support for artifact composition and relationship constraints in the GSM, our new approach aims to address this limitation by extending the current GSM semantics. The expected efforts to extend GSM with specific SE for AIS aspect include: (1) Covering the specific characteristics of SE for AIS (defining the suitable process modeling elements for managing data, feature engineering, training model etc.). (2) Defining novel relationships that enable the specification of AI-specific software and data artifacts at various levels of granularity, aiming to enhance traceability and ensure coherence. (3) Software artifacts might be define at different level of granularity, thus, it is essential to establish behavioral and temporal constraints.

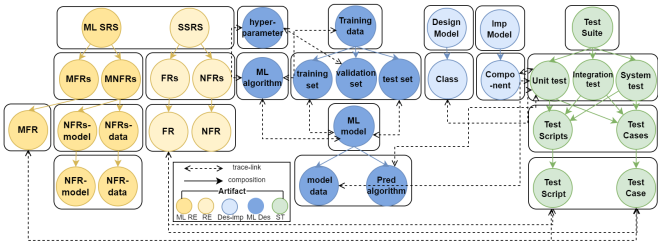

Fig. 3. Artifact-centric Macro process model for SE process for AIS with expected trace links between the artifacts.

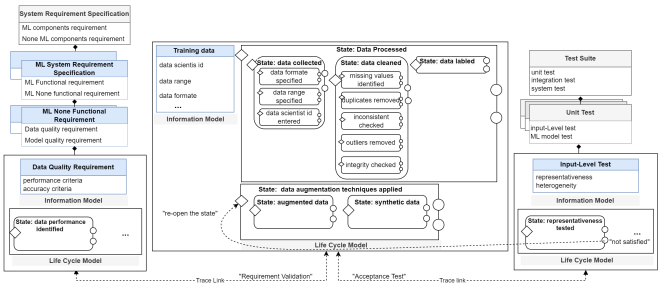

Fig. 4. Data Training artifact' micro process model modeled by Guard Stage Milestone process modeling language.

As depicted in Fig. 4 a Micro Process Model of Training data artifact consists of: (A) an *information model*: specifies the attributes relevant to describe the artifact and its relationships to other artifacts; And (B) The *life-cycle model* specifies the behavior of an artifact.

VI. Plan for Evaluation

We aim to provide a prototype for executing the process model including: Process Model Representation: define the language to represent the process model; Process Engine: implement a data-driven process engine to execute the process model; Process Data Storage: we aim to use a graph structure database (e.g., Neo4j); User Interface: develop a customized application user interface that facilitates the interaction with the process model. The evaluation metrics for the process model are contingent upon its alignment with process monitoring objectives. Presently, traceability serves as the primary goal, and we aspire to incorporate the following criteria: completeness, scalability and complexity.

VII. Current Status

The work completed to date include:
(1) Goal-oriented artifact-centric process modeling methodology; (2) Framework specifying the modeling characteristics of the new process modeling language for AIS.
The anticipated schedule for research completion is as follows: **Months 1-2**: data collection: identify and adequately define the specific characteristics for SE for AIS (including the specific list of artifacts, tasks, roles, dependencies and workflow). **Months 3-4-5**: Abstract syntax and structure of the DSL definition: specify the concepts, relationships, and constraints of the SPML4AI using a metamodeling language (expected tool to this task : Eclipse EMF); Semantics and Constraints of the DSL definition: (expected language: OCL (Object Constraint Language). **Months 6-7-8-9-10**: Implement the process engine: define the mechanisms for managing the process instances, including their lifecycle, state transitions, and data storage; Implement the required logic for controlling the flow of artifacts within a process instance; Implement the integration capabilities to interact with external systems; Implement mechanisms for handling errors.

VIII. Conclusion

We introduce *SPMNL4AI*, a framework designed to support the modeling and execution of Software Engineering (SE) for Artificial Intelligence Systems (AIS). The research investigates the effectiveness of current Process Modeling Languages (PMLs) in light of the new demands posed by ML components development process. To date, there is a lack of PMLs specifically tailored to address the challenges presented by SE for AIS, which require certain essential requirements to be supported by PMLs. These requirements include the data-centric aspect and flexibility necessary for ML training and tuning development processes. In this paper, we propose new characteristics that must be supported by PMLs to effectively model SE processes for AIS. These enhancements are crucial to accommodate the unique demands and complexities of AI development and foster more efficient and accurate SE practices in the context of AIS.

REFERENCES

[1] J. Lonchamp, "A structured conceptual and terminological framework for software process engineering," 04 2001.

[2] S. Amershi, A. Begel, C. Bird, R. Deline, H. Gall, E. Kamar, N. Nagappan, B. Nushi, and T. Zimmermann, "Software engineering for machine learning: A case study," pp. 291–300, 05 2019.

[3] H. Villamizar, T. Escovedo, and M. Kalinowski, "Requirements engineering for machine learning: A systematic mapping study," in *2021 47th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 29–36, 2021.

[4] J. Bosch, H. Olsson, and I. Crnkovic, *Engineering AI Systems: A Research Agenda*, pp. 1–19. 01 2021.

[5] D. Sculley, G. Holt, D. Golovin, E. Davydov, T. Phillips, D. Ebner, V. Chaudhary, M. Young, J.-F. c. Crespo, and D. Dennison, "Hidden technical debt in machine learning systems," in *Advances in Neural Information Processing Systems* (C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds.), vol. 28, 2015.

[6] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella, "Testing machine learning based systems: a systematic mapping," *Empirical Software Engineering*, vol. 25, pp. 5193–5254, 2020.

[7] P. Kriens and T. Verbelen, "What machine learning can learn from software modularity," *Computer*, vol. 55, no. 9, pp. 35–42, 2022.

[8] K. Ahmad, M. Abdelrazek, C. Arora, M. Bano, and J. Grundy, "Requirements engineering for artificial intelligence systems: A systematic mapping study," *Information and Software Technology*, vol. 158, p. 107176, 2023.

[9] Z. Pei, L. Liu, C. Wang, and J. Wang, "Requirements engineering for machine learning: A review and reflection," *2022 IEEE 30th International Requirements Engineering Conference Workshops (REW)*, pp. 166–175, 2022.

[10] K. Habibullah, G. Gay, and J. Horkoff, "Non-functional requirements for machine learning: understanding current use and challenges among practitioners," *Requirements Engineering*, vol. 28, 01 2023.

[11] J. Horkoff, "Non-functional requirements for machine learning: Challenges and new directions," *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pp. 386–391, 2019.

[12] S. K. Singh and A. Singh, *Software testing*. Vandana Publications, 2012.

[13] OMG, "Software systems process engineering meta-model specification, version 2.0 - omg document formal/2008-04-01."

[14] OMG, "Business process model and notation, version 2.0 - omg document formal/2011-01-03."

[15] R. Abualsaud, H. Tran, I. Ober, and M. Nguyen, "Toward a goal-oriented methodology for artifact-centric process modeling," 2023.

[16] R. Hull, "Artifact-centric business process models: Brief survey of research results and challenges," in *On the Move to Meaningful Internet Systems: OTM 2008* (R. Meersman and Z. Tari, eds.), (Berlin, Heidelberg), pp. 1152–1163, Springer Berlin Heidelberg, 2008.

[17] V. Künzle and M. Reichert, "Philharmonicflows: Towards a framework for object-aware process management," *Journal of Software Maintenance and Evolution Research and Practice*, vol. 23, pp. 205–244, 06 2011.

[18] J. Kunchala, J. Yu, and S. Yongchareon, "A survey on approaches to modeling artifact-centric business processes," vol. 9051 of *Lecture Notes in Computer Science*, pp. 117–132, Springer, 2014.

[19] R. Hull, E. Damaggio, R. D. Masellis, F. Fournier, M. P. Gupta, F. T. Heath, S. F. Hobson, M. H. Linehan, S. Maradugu, A. Nigam, N. Sukaviriya, and R. Vaculín, "Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events," in *Distributed Event-Based Systems*, 2011.

[20] OMG, "Case management model and notation, version 1.1 - omg document formal/2016-10-24."

[21] W. Aalst, G. Li, and M. Montali, "Object-centric behavioral constraints," 03 2017.