DevOpsML: Towards Modeling DevOps Processes and Platforms

Alessandro Colantoni
Institute of Business Informatics - Software Engineering
Johannes Kepler University
Linz, Austria
alessandro.colantoni@jku.at

Luca Berardinelli
Institute of Business Informatics - Software Engineering
Johannes Kepler University
Linz, Austria
luca.berardinelli@jku.at

Manuel Wimmer
Institute of Business Informatics - Software Engineering
Johannes Kepler University
Linz, Austria
manuel.wimmer@jku.at

ABSTRACT

DevOps and Model Driven Engineering (MDE) provide differently skilled IT stakeholders with methodologies and tools for organizing and automating continuous software engineering activities–from development to operations, and using models as key engineering artifacts, respectively. Both DevOps and MDE aim at shortening the development life-cycle, dealing with complexity, and improve software process and product quality.

The integration of DevOps and MDE principles and practices in low-code engineering platforms (LCEP) are gaining attention by the research community. However, at the same time, new requirements are upcoming for DevOps and MDE as LCEPs are often used by non-technical users, to deliver fully functional software. This is in particular challenging for current DevOps processes, which are mostly considered on the technological level, and thus, excluding most of the current LCEP users. The systematic use of models and modeling to lowering the learning curve of DevOps processes and platforms seems beneficial to make them also accessible for non-technical users.

In this paper, we introduce DevOpsML, a conceptual framework for modeling and combining DevOps processes and platforms. Tools along with their interfaces and capabilities are the building blocks of DevOps platform configurations, which can be mapped to software engineering processes of arbitrary complexity. We show our initial endeavors on DevOpsML and present a research roadmap how to employ the resulting DevOpsML framework for different use cases.

CCS CONCEPTS

• **Software and its engineering → Abstraction, modeling and modularity**; **Integration frameworks**.

KEYWORDS

DevOps, model-driven engineering, modeling languages

ACM Reference Format:
Alessandro Colantoni, Luca Berardinelli, and Manuel Wimmer. 2020. DevOpsML: Towards Modeling DevOps Processes and Platforms. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3417990.3420203

1 INTRODUCTION

Over the last decade, DevOps methods and tools have been successfully implemented and adopted by companies to boost automation and efficiency of the engineering process. The term DevOps was coined in 2009 [7] and became popular among companies and practitioners [25] and, subsequently, among researchers and academia. Jabbari et al. [23] define DevOps as *"[...] a development methodology aimed at bridging the gap between Development and Operations, emphasizing communication and collaboration, continuous integration, quality assurance and delivery with automated deployment utilizing a set of development practices."*.

The momentum on DevOps resulted in a flourishing of technological solutions to meet the huge market demands [22]. The side effects of such a rapid evolution was a scattered landscape of technological solutions offering a variety of tools [11] for supporting activities of *continuous-software engineering* (CSE) [20] processes.

Figure 1 gives a bird's eye view of the problem at hand. There is no "one size fits all" DevOps process that is capable to cope with all the specific goals, strategies, and requirements. Different DevOps process variants exist (e.g., DevSecOps [28] or AIOps [15] to mention just a few). The process variability is reflected on DevOps platforms too, which may aggregate different engineering services (e.g., security mechanisms and AI-augmented services) depending on process needs.

Consequently, the choice of DevOps platforms for specific engineering processes is still an open challenge. In [7], Bordelau et al. investigated and elicited sets of requirements for DevOps frameworks. Among them, they consider also the need for an adequate support for modeling of DevOps engineering *processes*, of the *product* resulting from the process, i.e., the software system, as well as requirements of *resources* (e.g., tools) involved in the accomplishment of development and operations phases. Furthermore, it has to be mentioned that the same DevOps process and platform can be more or less adequate based on different skills of the involved stakeholders, nowadays possibly ranging from skilled engineers to domain experts with no ICT background at all.

When the term DevOps was coined, principles and practices of Model Driven Engineering (MDE) were already spread among researchers and practitioners [4, 8], and it is now consolidating its own body of knowledge [10]. In MDE, *models* are considered the keystones of any engineering activity, from application-level ones at the highest level of conceptualization down to system implementation, deployment, and operations. Models are prescriptive, machine-readable artifacts, obtained as results of modeling activity

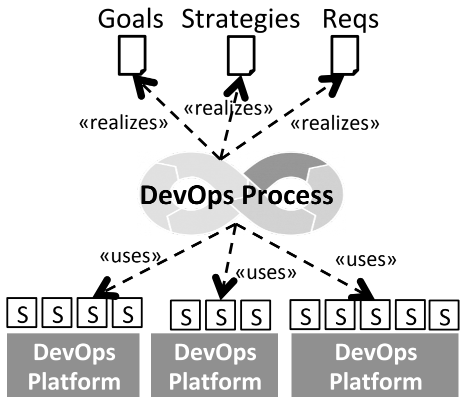


Figure 1: Weaving together DevOps processes and platforms.

and the software is the result of a (semi-)automated chain(s) of *transformations* among interwoven, validated models [8]. While MDE has been applied extensively for generating software products, it has been less frequently used for generating infrastructure code required to run continuous engineering processes such as DevOps.

In order to tackle this shortcoming, we introduce in this paper DevOpsML, a conceptual framework for modeling and configuring DevOps engineering processes and platforms. With DevOpsML, we approach the problem of DevOps process and platform integration from two directions. First, in a bottom-up approach, existing DevOps platforms can be studied and their characteristics made explicit in so-called platform models. Second, in a top-down approach, we propose the usage of process modeling languages [21] to explain the DevOps processes. The glue between the two directions is a linking language, which explains how the different services offered the platforms are used by the processes.

We outline the DevOpsML framework with all its main constituents, show a typical application for a given scenario, and present a roadmap on how DevOpsML may be *extended* and used in the future for different use cases. In particular, since DevOpsML is conceived in the context of the Lowcomote EU project [36], we expect to study its application in the context of low-code engineering platforms (LCEP)[1], as the MDE-wise evolution of *low code development platforms* (LCDPs) [36].

The rest of the paper is organized as follows: Section 2 details the problem addressed by this paper and introduces the DevOpsML framework. Section 3 presented some guidelines how to use DevOpsML and demonstrates the language by-example. Section 4 proposes a roadmap for future research directions inspired by DevOpsML, in particular, for which use cases it may be employed. Finally, Section 5 discusses related work, while Section 6 concludes the paper.

2 THE DEVOPSML FRAMEWORK

This section introduces the DevOpsML framework and its three main components, i.e., platform specification, software process specification, integration mechanism in a technology-agnostic manner.

[1]See *ESR 9: DevOps Support for Low-Code Engineering Platforms* at https://www.lowcomote.eu/esr/09/

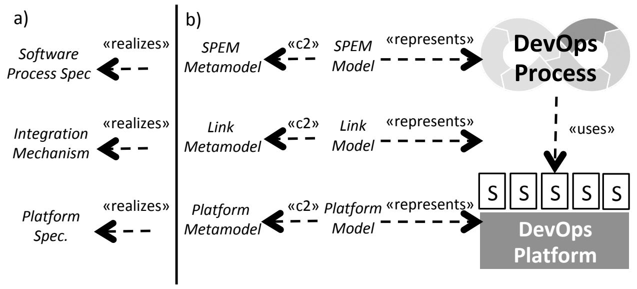


Figure 2: The DevOpsML framework elements (a) and their implementations (b).

Subsequently, we propose a preliminary implementation of DevOpsML based on model-driven technologies offered by the Eclipse EMF [19] and Epsilon [24].

2.1 DevOpsML at a Glance

In order to provide a systematic and integrated view on DevOps processes and their deployment to services provided by DevOps platforms (cf. Figure 1), we introduce in this paper DevOpsML, a conceptual framework (cf. Figure 2) for modeling and configuring DevOps engineering processes and platforms. First, DevOpsML allows defining so-called platform models that describe the different DevOps platforms as well as their offered services by proposing a platform modeling language. Second, DevOpsML allows defining explicit engineering processes. For this purpose, we make use of the modeling standard SPEM. In order to map the different processes to the platforms, we provide a weaving mechanism that integrates, via inter-model links, process and platform specifications.

In the following, we detail each part of DevOpsML by describing the main modeling concepts employed for our purposes.

2.2 Platform Specification

According to the given DevOps definition [23], we expect that a platform configuration is capable to support development and operational activities adopting and realizing MDE principle and practices [8].

In DevOpsML, a platform is a combination of tools, each one providing (or requiring) a set of capabilities addressing engineering concerns of interest, supported by combining tools via explicit interfaces. A platform is expected to satisfy and being able to satisfy the needs of an arbitrary engineering process.

Following typical MDE practices, we provide a *platform metamodel* to create platform models representing a combination of tools, interfaces, capabilities, and concerns. The metamodel with its concepts and relationships is shown in Figure 3 and implemented in Ecore [5]. The following paragraphs detail the content of the platform metamodel.

Tools and Interfaces. This package introduces metaclasses for representing tools and interfaces.

In its broadest meaning, a tool is a resource that helps in accomplishing a work unit of an engineering process. It provides or requires interfaces.

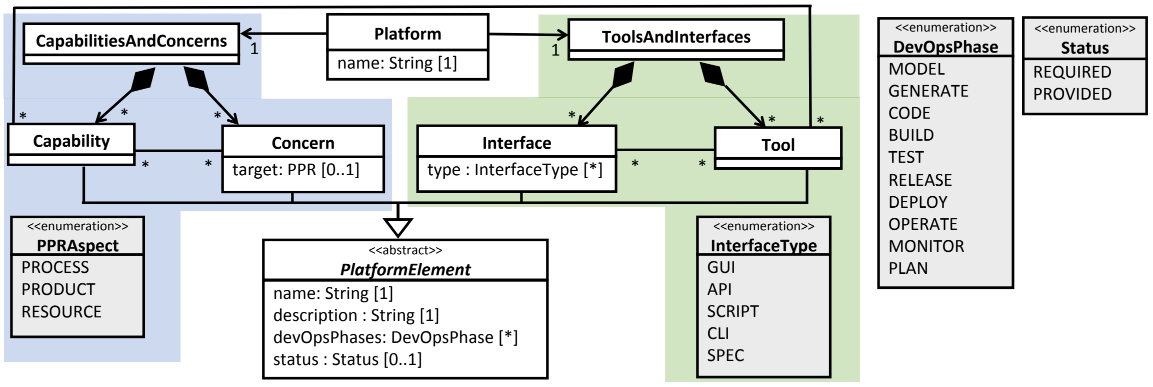


Figure 3: Platform metamodel.

An interface represents the boundary of tools, and consequently, of platforms as a whole, through which interactions among tools and platforms take place, according to their required and provided capabilities. Interfaces play the roles of connectors [32] among platform elements, directly connecting multiple tools and, through them, their capabilities. We consider a predefined but extensible set of interface types, i.e., graphical user interfaces (GUI), application programming interfaces (API), script, command line interfaces (CLI), or more generic specifications (SPEC).

Capabilities and Concerns. This package introduces the concepts of *capability* and *concern.*

For *capability*, we intend a facility enabled by a particular platform configuration for performing a specified activity that will be specified in a separated process model (see Section 2.3).

A *concern* is *"a stakeholder's interest that pertains to the development of an application, its operation or any other matters that are critical or otherwise important"* [37].

A platform configuration is expected to offer capabilities to address concerns related to (*i*) the engineering process, (*ii*) the system under study, i.e., the *product* of the engineering process, and (*iii*) of the *resources* (both human and technological ones) required for the correct and convenient process execution and delivery of the engineered product. We introduce the *PPRAspect* enumeration to distinguish among process, product, and resource concerns [34].

Common concepts. The platform metamodel also includes some shared concepts: platform element, status, and DevOps phase.

A platform element is an abstract concept that groups common properties of capabilities, concerns, tools, and interfaces. For all these platform elements, a name and a textual description are mandatory and, together, correspond to the minimal wealth of knowledge required to model platform elements.

Since we intend to model DevOps platforms, we expect that its elements will support typical DevOps process phases (code, build, test, release, deploy, operate, monitor, and plan), which are given as literals of the DevOps phase enumeration. In addition, we aim at applying MDE principles and practices and for this reason, we include model and generate phases, which are recurrent activities in model-driven engineering processes. Each platform element can be associated with many DevOps phases.

It is worth nothing that the given DevOps phase enumeration is not exhaustive and it can be extended, if needed. However, we expect detailed engineering process information to be modeled in separated process models (see Section 2.3). Finally, provided or required status can be set for any platform element, providing rationales for composing the platform like technology-wise matching of required and provided tools' interfaces or higher-level evaluation of platforms' capabilities against engineering process or product-related concerns.

2.3 Process Specification

Process management is a core concern for software engineering since decades [9, 17] and regards the specification and execution of organizational behaviours, where working units at different level of granularity are combined in a workflow. Stakeholders with defined roles collaborate to perform the process. Engineering artifacts are produced and manipulated throughout the process.

In DevOpsML, a platform is meant to support model-driven CSE processes, where activities are expected to manipulate and share MDE artifacts [16], aiming at the highest degree of automation.

In DevOpsML, we assume that a process (model) provides the rationales to choose the elements of a DevOps platform (model), defined as described in Section 3.3. For the sake of process specification, DevOpsML needs a software process modeling language (SPML). In [21], a quality model for SPMLs is given.

For our first prototypical implementation of DevOpsML [5], we choose the Software and Systems Process Engineering Metamodel (SPEM) [29]. SPEM satisfies a minimal set of modeling capabilities that we require for a DevOpsML proof of concept phase. Table 1 provides a list of common capabilities of process modeling languages described in [1] for the sake of a process modeling challenge[2] and maps them to the corresponding concepts defined in SPEM [29]. The complete mapping is available in [5].

Second, we decide to give higher importance to the descriptive capability of SPEM [9] for documentation purposes rather than executability, in which case BPMN or UML Activities (via Foundational UML (fUML) [30]) are more appropriate solutions than SPEM. Moreover, SPEM supports the specification of new processes by

[2]The table report in parenthesis the original IDs (Px) of the process modeling capabilities.

Table 1: Process modeling capabilities [1] and their support in SPEM

Process modeling capability	**SPEM (MC:Method Content Package, PM:Process with Methods Package)**
A process type is defined by the composition of one or more task types. Each process comprises one or more tasks (P1, P11).	PM:Activity; PM:Task Use
Each task type is created by an actor. An actor may have more than one actor type. An actor that performs a task must be authorized for that task. Actor types may specialize other actor types (P4, P15,P17, P18).	MC:Role Definition; PM:Composite Role; PM:Role Use; PM:Team Profile
Tasks are associated with artifacts used and produced, along with performing actors. For each task type one may stipulate the artifact types which are used and produced (P7, P13, P14)	MC:Task Definition; MC:Work Product Definition; MC:Role Definition; MC:Default Task Definition Parameter; PM:Work Product Use;

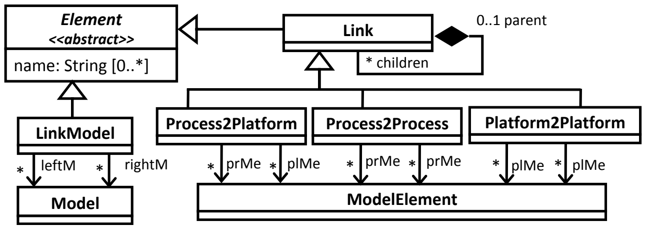


Figure 4: The Linking metamodel.

separating the definition of reusable process model elements (see Method Content language package in [29]) and their actual uses in processes (see the Process with Methods language package in [29]). In Table 1, we reported the owning package of each mapped concept that we use later in Section 3.1 to show DevOpsML in action.

However, it is worth noting that the choice of a particular SPML for process specification is a variation point of DevOpsML. Indeed, different SPML can be chosen from existing ones [21] or new ones can be created following software language and model-driven engineering practices [10, 13] to cope with arbitrary SPML's requirements like process modeling capabilities [1] or usability by (non-)technical users.

2.4 Integration Mechanism

DevOpsML is a model-driven framework and, as such, different model integration mechanisms are good candidates to provide a model integration capability, such as model weaving and model transformation [8].

For DevOpsML, we choose the model weaving mechanism to link elements from platform and process models. Epsilon Modelink [24] for establishing and editing references between platform and process models.

Figure 4 shows a *linking metamodel* for this purpose. A *link model* contains a collection of hierarchical *links*. Since we do not specify any direction for links, we consider them bidirectional by construction. Each link is weaving sets of model elements belonging to process or platform models[3].

[3]We keep the Linking metamodel simple on purpose. We do not show here the specialisation of ModelElement metaclass and constraints to distinguish intra/inter-model links.

Based on the nature of the woven models, three different types of links can be distinguished: *platform to process*, *platform to platform*, and *process to process*:

- **Process to Platform**: process work units, at any granularity level (e.g., SPEM activities, tasks, steps) can be mapped to platform elements (tools, interfaces, capabilities, concerns).
- **Platform to Platform**: elements from two or more platform models can be linked for user-defined rationales. For example, based on values set for status property, potential (mis)matches can be modeled and used for evaluations (e.g., classification of different platform configurations against concerns' coverage criteria).
- **Process to Process**: elements from two or more process models can be linked for user-defined rationales. In particular, since DevOpsML does not prescribe the use of any process modeling language, these links can be used to relate process models defined with different process modeling languages[4].

3 A TOUR ON DEVOPSML

This section provides a guideline on the usage of the DevOpsML framework and shows it in action on explanatory examples. An activity-like workflow is sketched in Figure 5. It comprises four modeling activities (rounded boxes), whose outcomes are (i) process model(s), (ii) platform model(s), (iii) libraries of reusable platform elements (tools, interfaces, capabilities, concerns) to be reused for platform modeling, and (iv) linking model(s) combining processes and platform models. The final outcome is a DevOps model, a tripartite model including the artifact produced throughout the proposed workflow.

The next sections explain the four activities in details. All the mentioned artifacts are available and further detailed in the project repository [5].

3.1 Process modeling

In order to perform the process modeling activity, of course, we need a process metamodel (*Process MM*) and a corresponding process model editor.

We split the process modeling activities in two sub steps: *initial process modeling* and *process refinement*. A first version of a *process model* is created by collecting a set of coarse-grained working

[4]Technical limitation to this scenarios apply if the integration mechanism does not support heterogeneous modeling spaces [16].

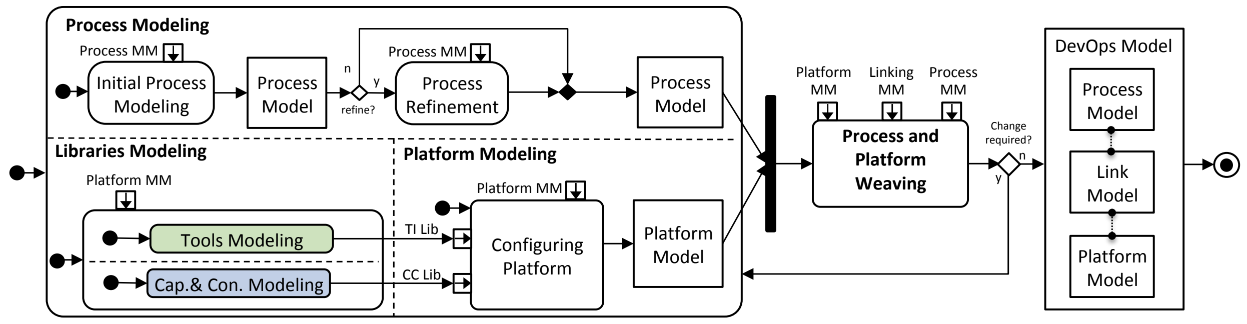


Figure 5: Guidelines for process and platform modeling, and their integration.

units and organizing them in a workflow. The process model may suffice (or not) the needs of the involved stakeholders. If needed, a *process refinement* step takes place, where a process model can be further enriched with details like fine-grained tasks, artifacts, tools descriptions, and responsibilities.

Example. We choose the OMG SPEM [29] as process modeling language for the reasons aforementioned in Section 2.3. We create a SPEM process model as instance of a SPEM metamodel, which can be edited via metamodel-driven editors (e.g., tree-based editors automatically generated via Eclipse EMF) or via UML editors equipped with the SPEM UML profile [29] such as MagicDraw[5].

For the sake of illustration purposes, we show the content of SPEM model as created with the MagicDraw SPEM editor while its EMF-based version is available in [5]. Figure 6a represents a coarse-grained DevOps-like process on a SPEM workflow diagram. The SPEM activities span development (plan, model/code, build, test, release, deploy) and operation phases (operate, monitor) and they can be continuously repeated until the system under study (SUS) is running. It is a simplified view of a typical DevOps process where modeling and coding activities could be combined (e.g., via model to code transformations) [14]. Such coarse grained process model does not provide further details.

Process refinements can be applied on such initial process model. We illustrate this step by hierarchically combining the initial workflow in Figure 6a with an additional SPEM workflow and a set of activity detail diagrams (ADD) as shown in Figure 6b.

An ADD describes the internal structure of an activity with guidance (as textual description), definitions and occurrences of tasks, roles, work products, tools, and their relationships.

In Figure 6b, we modeled the release strategy proposed as *Gitflow Workflow* by Atlassian [2] using concepts taken from the SPEM Process with Methods and Method Content packages (see Table 1) [29].

Following the SPEM modeling guideline, we distinguish between *definitions* and occurrences or *uses* of process model elements. Therefore, we refined the *release* activity with a workflow made of four task uses, namely *start*, *perform*, *merge* and *delivery* and provide details via separated ADDs. Each ADD refines a task use (i) by modeling related inputs/outputs artifacts (a.k.a. work product uses) and role responsibilities, and (ii) the corresponding reusable work product and tasks definitions, together with tool definitions required to accomplish the given task.

[5]https://www.nomagic.com/product-addons/no-cost-add-ons/spem-plugin

In particular, the task *start* takes in input the *develop* branch and forks it to the *release* branch. The *branches operation* requires *source code manager* and *CI-CD-Server* tools to manage a *branch*.

The task *perform* takes in input the created *release* branch and uses it to polish the release, fixing bugs, generating documentation, and for other release-oriented tasks, not shown in Figure 6 due to space limitation. The output is the *release product package*. The task *perform* is an occurrence of a *release operation* that requires *build manager* and *CI-CD-Server* tools to manage a *Package*.

The next task *merge*, once *perform* is completed, merges the *release* branch into the *master* and *develop* branches. A tag with the version number is then created. In SPEM, both start and merge are modeled as Task Uses of the same *branches operation* Task Definition.

The *delivery* task takes in input the *release product*. According to the *delivery operation* Task Definition, its accomplishment requires an *Artifact Repository* tool where each work product, as package, is collected.

The outcome of the process modeling activity is a process model whose complexity depends on the inherent complexity of the considered engineering process, the modeling needs and expertise of the process modeler. It is worth noting that, depending on the expressiveness of the chosen SPML, platform-related information can included as well at this stage (e.g., SPEM tool definitions).

3.2 Libraries Modeling

A platform model is the combination of tools, interfaces, concerns and capabilities. All platform elements can be modeled from scratch, thus requiring a huge modeling effort. For this reason, we expect to collect platform elements in reusable *libraries*.

The platform metamodel is designed on purpose with no containment references from Platform to Tools & Interfaces (TI) and Capabilities & Concerns (CC) metaclasses. The result of this metamodel design choice is the possibility to provide two libraries of platform elements in separated models, Indeed, we expect the creation and population of two distinct libraries (TI Lib and CC Lib in Figure 5) as instances of Tools & Interfaces and Capabilities & Concerns metaclasses, respectively.

Those TI and CC libraries can then be populated by different providers, which can evolve them independently from each other and from any platform configuration. Tool providers, practitioners, and researcher can model their own or preferred industrial-strength

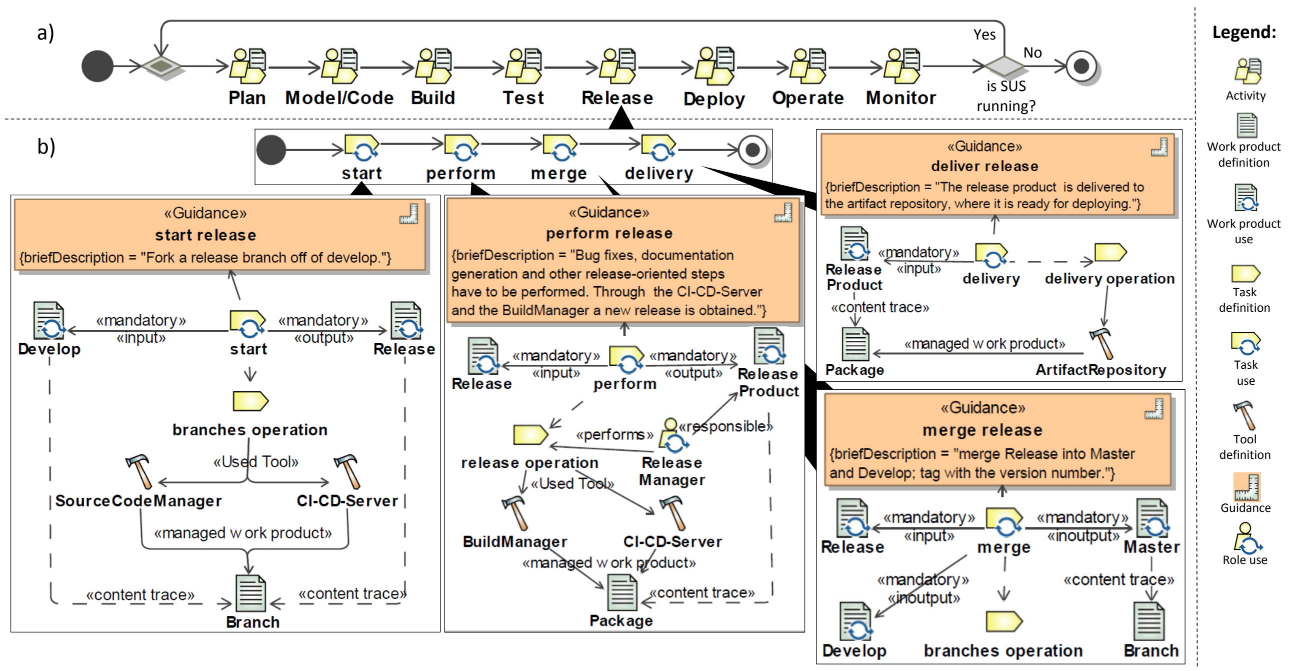


Figure 6: Process modeling with SPEM: a) coarse-grained DevOps-like engineering process and b) its refinement.

and research tools and interfaces. At the same time, stakeholders involved in the engineering process can model their concerns of interest. For example, required concerns and required capabilities can be used together to model process, product, or resource requirements, which can be matched with provided capabilities. Whenever possible, capabilities can be linked to tools collected in corresponding libraries, thus providing a concrete aid for platform modeling and weaving with provided process models.

Moreover, we expect to use and co-evolve libraries (i.e., both their content as well as the corresponding metaclasses in Figure 3) to represent DevOps variants, technology-specific information (e.g., cloud concepts such as auto-scaling, vendor-specific and -agnostic cloud APIs, containerisation and run-time standards) that we expect from a fast-paced, evolving domain like DevOps.

The platform elements shows in Figure 7 can be considered the very first entries of TI and CC libraries. We are building from scratch the first versions of such platform libraries and they are continuously updated and available on the project repository [5].

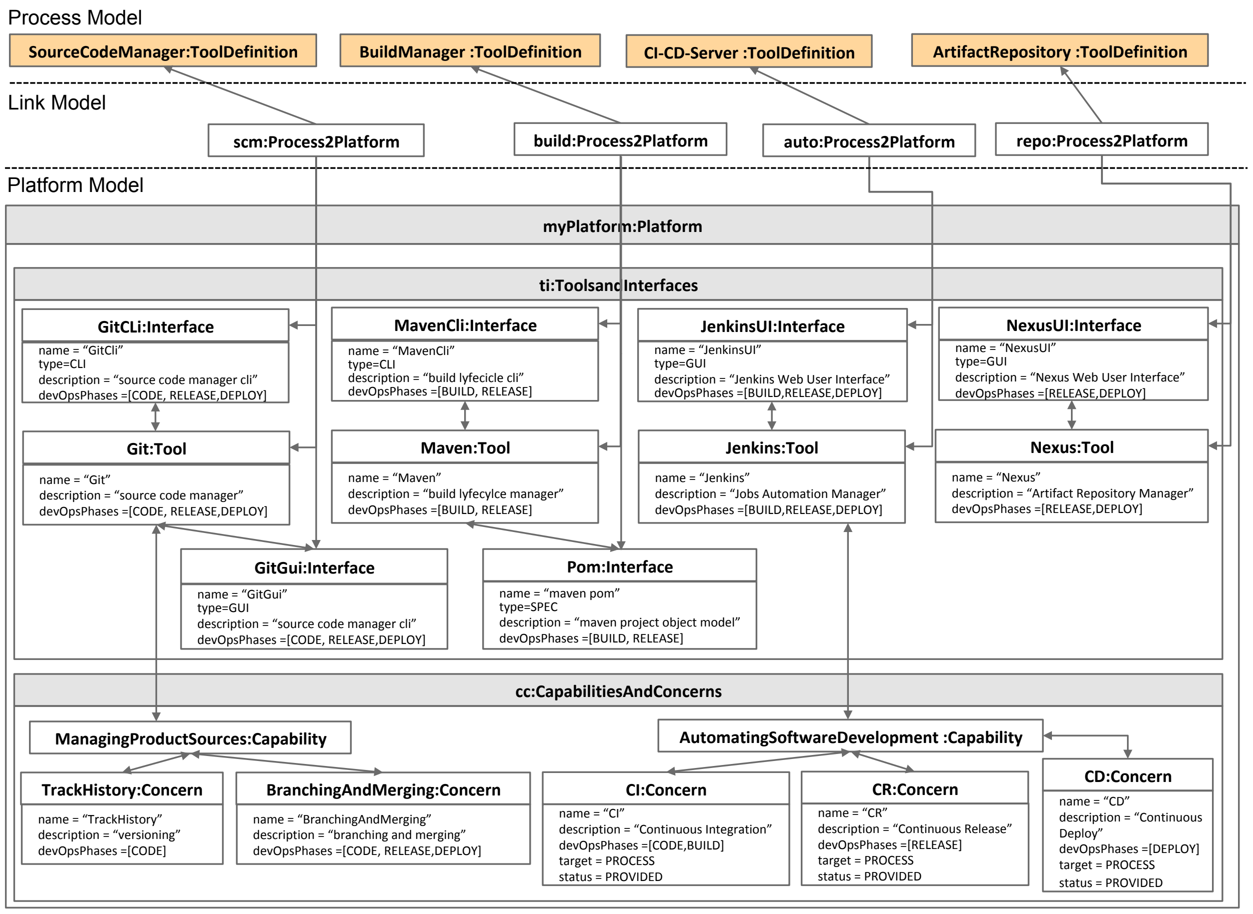
3.3 Platform modeling

Platform modeling consists in the selection and combination of platform elements in a platform model compliant to the metamodel in Figure 3. At this stage, the engineering process can be partially known or completely unknown. Process knowledge sources are (*i*) high level, partial or detailed process models (see Section 3.1), and (*ii*) process concerns (i.e., concerns with target property set to process).

Based on the mix of available process information, and assuming the availability of platform elements from TI and CC libraries, a platform modeler can create a platform model by iterative refinements . In the following, we describe two complementary criteria for selecting platform elements, sketching the metamodel-level navigation scheme (in parenthesis):

- **Tool-first criterion**: Existing tools and interfaces are selected that provide capabilities to support one or more DevOps phase (e.g., *tool⟶capability⟶concern:target=process, devOps phases= any*). This criterion can be followed by tool providers that want to model their platform to be offered on the market.
- **Required capability-first criterion**: Existing required process concerns are taken into consideration as process requirements. Sets of candidate tools are selected that provide one or more of the required capabilities (e.g., *concern:target=process, status=required ⟶capability* matching *tool⟶capability: status=provided*). This criterion can be applied to model a desired DevOps platform.

In Figure 7, a platform model is depicted for explanatory purposes using an object-like diagram notation. The modeled platform addresses five process concerns (track history, branching and merging, continuous integration, continuous release, and continuous deploy) that are linked to two capabilities (merging product sources, automating software delivery) provided by two tools, Git and Jenkins, respectively, via Git command line interface (GitCLI) and Jenkins graphical user interface (JenkinsUI). The platform model is then completed by the artifact repository manager Nexus, and the build manager Maven and the corresponding interfaces Nexus UI and Maven CLI.



Process Model
SourceCodeManager:ToolDefinition
BuildManager :ToolDefinition
CI-CD-Server :ToolDefinition
ArtifactRepository :ToolDefinition
Link Model
scm:Process2Platform
build:Process2Platform
auto:Process2Platform
repo:Process2Platform
Platform Model
myPlatform:Platform
ti:ToolsandInterfaces
GitCLi:Interface
name = "GitCli"
type=CLI
description = "source code manager cli"
devOpsPhases =[CODE, RELEASE,DEPLOY]
MavenCli:Interface
name = "MavenCli"
type=CLI
description = "build lyfecicle cli"
devOpsPhases =[BUILD, RELEASE]
JenkinsUI:Interface
name = "JenkinsUI"
type=GUI
description = "Jenkins Web User Interface"
devOpsPhases =[BUILD,RELEASE,DEPLOY]
NexusUI:Interface
name = "NexusUI"
type=GUI
description = "Nexus Web User Interface"
devOpsPhases =[RELEASE,DEPLOY]
Git:Tool
name = "Git"
description = "source code manager"
devOpsPhases =[CODE, RELEASE,DEPLOY]
Maven:Tool
name = "Maven"
description = "build lyfecylce manager"
devOpsPhases =[BUILD, RELEASE]
Jenkins:Tool
name = "Jenkins"
description = "Jobs Automation Manager"
devOpsPhases =[BUILD,RELEASE,DEPLOY]
Nexus:Tool
name = "Nexus"
description = "Artifact Repository Manager"
devOpsPhases =[RELEASE,DEPLOY]
GitGui:Interface
name = "GitGui"
type=GUI
description = "source code manager cli"
devOpsPhases =[CODE, RELEASE,DEPLOY]
Pom:Interface
name = "maven pom"
type=SPEC
description = "maven project object model"
devOpsPhases =[BUILD, RELEASE]
cc:CapabilitiesAndConcerns
ManagingProductSources:Capability
AutomatingSoftwareDevelopment :Capability
TrackHistory:Concern
name = "TrackHistory"
description = "versioning"
devOpsPhases =[CODE]
BranchingAndMerging:Concern
name = "BranchingAndMerging"
description = "branching and merging"
devOpsPhases =[CODE, RELEASE,DEPLOY]
CI:Concern
name = "CI"
description = "Continuous Integration"
devOpsPhases =[CODE,BUILD]
target = PROCESS
status = PROVIDED
CR:Concern
name = "CR"
description = "Continuous Release"
devOpsPhases =[RELEASE]
target = PROCESS
status = PROVIDED
CD:Concern
name = "CD"
description = "Continuous Deploy"
devOpsPhases =[DEPLOY]
target = PROCESS
status = PROVIDED

Figure 7: Weaving a SPEM process and platform model elements via Link Model.

3.4 Process and Platform Weaving

The process and platform weaving activity combines process and platform models obtained from the corresponding modeling activities. Both platform and process models are now considered as consolidated. Weaving links are created using the integration mechanism introduced in Section 2.4.

A Link Model is shown in Figure 7 using an object diagram-like notation. The links shown in this example are set among multiple tools (Git, Maven, Jenkins, Nexus) from the platform model introduced in Section 3.3 and SPEM tool definitions depicted on the SPEM ADD described in Section 3.1. It is worth noting that elements from process and platform models potential are valid ends for process to platform links. It is up to the modeler and her experience, to bind the correct elements from the two models. We currently do not provide assistance for this task and we leave it as future work to provide more modeling intelligence.

A possible goal of the weaving process and of process to platform links is assessing the suitability of the given platform (model) in supporting the execution of the given process (model), and vice versa.

A positive assessment results in a tripartite DevOpsML model can be reused as input to any model-driven engineering activity. In this paper, we do not pose any further validation criteria for the resulting DevOpsML Model and is left for future work.

In case of a negative assessment of the resulting DevOpsML model, the modeler can go back and resume previous activities (process, library, and platform modeling) depending on the cause of the unsatisfactory outcome (e.g., poorly designed process, platform elements, or platform as a whole).

4 DEVOPSML ROADMAP

In this section, we highlight some future work on DevOpsML and outline associated research directions.

Languages as platform elements. Languages are typical instruments in the hand of engineers to specify artifacts. In [8], modeling languages are defined as *"conceptual tools aimed at letting designers formalize their thoughts and conceptualize the reality in explicit form, being it textual or graphical"*. In addition to modeling languages, DevOps platforms can be also described in terms of programming and scripting languages used to implement the integrated APIs. In addition, JSON and YAML-based languages are common solutions for DevOps platform configuration concerns.

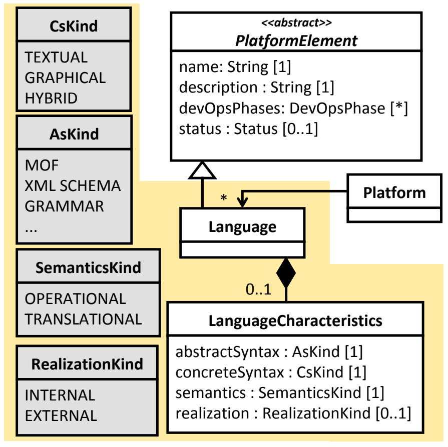


Figure 8: Languages as new platform elements.

For this reasons, we aim at including *languages* as first class DevOps platform elements. By raising languages to the status of platform elements, we may better assess the application of MDE principles and practices [16] by different DevOps platforms. Figure 8 shows an extension of the platform metamodel in Figure 3.

A language is a new platform element, which can be further detailed via language characteristics such as abstract syntax (*AsKind*), concrete syntax (*CsKind*), *Semantics*. Details about language's technical realization can also be provided, for example whether it is an independent or embedded language in host language (*RealizationKind*). A modeling language is expected to provide such information in order to determine the modeling space of $l \in L$ [4, 16]. Since part of the success of DevOpsML will also depend on the availability of libraries of reusable platform elements, we will consider the introduction of a language library and its proper use within the workflow sketched in Figure 5.

DevOpsML for Low-Code Engineering Platforms. DevOpsML is conceived in the context of the Lowcomote EU project [36]. For this reason, we envision its first adoption the configuration of low-code engineering platforms (LCEP)[6].

LCEPs are meant to be used by non-technical users or *citizen developers*, to deliver fully functional software, with the promise of automating and reducing the effort of time-consuming and complex engineering activities, like coding. Through advanced graphical user interfaces, visual mechanisms and declarative languages, citizen developers can focus only on the solutions, applying their knowledge at the right level of abstraction in the domain of their expertise [33, 36].

From a DevOpsML perspective, LCEP can be represented as platform model(s). Moreover, due to their particular nature, additional requirements need to be considered in order to keep DevOpsML accessible and usable to citizen developers. For example, simpler SPMLs can be used in place of SPEM for process specification, which may (not) be provided by a particular LCEP. Finally, special requirements can be considered for (modeling) languages to be suitable for LCEP such as hybrid notations [12].

[6]See *ESR 9: DevOps Support for Low-Code Engineering Platforms* available at https://www.lowcomote.eu/esr/09/

Reusable libraries for different needs. The DevOps model obtained as output of the workflow in Figure 5 can reuse platform elements collected in TI and CC libraries. We expect that their content will undergo continuous improvements that may include technological details to make them (and the resulting DevOps platform) suitable for specific purposes. In particular, libraries can be populated with MDE, security, or AI-augmented tools and APIs offering different capabilities (e.g., model management, security verification, or data analysis) for DevOps processes and platforms as expected by DevOpsML users playing different roles (e.g., modeler, security expert, data scientist).

DevOpsML usage scenarios. In this paper, we introduced a prototypical version DevOpsML and with the explicit intent to support the documentation of DevOps processes and platforms via distinct yet interwoven MDE artifacts (process model, platform model, link model). For this purpose, we chose SPEM as the most suitable SPML.

We plan to investigate more complex usage scenarios for DevOpsML, which may include (but it is not limited to) runtime platform and process monitoring, traceability, recovery, or simulation just to mention a few. Therefore, we plan to collect and study additional requirements for DevOpsML like executability of process and platform models or mechanisms to suitably propagate and combine design-time and operational data [27].

Service-orientation. A DevOpsML framework model represents a software engineering platform, providing and requesting (modeling) capabilities built on top of integrated tools and languages are integrated to build software products [34]. As future work, we plan to extend DevOpsML to support the description of capabilities *as a service* [31], as depicted in Figure 2. It is a matter of discussion whether the service-orientation concern should result in a evolved platform metamodel (e.g., by introducing a Service metaclass) or by extending libraries with service-oriented platform elements.

DevOpsML implementation. The current prototypical implementation of DevOps is based on Eclipse-based technologies i.e., EMF [19] for (meta) modeling. In this regard, we will investigate the use of MDE frameworks to support model management activities (e.g., model validation and transformation support). We will discuss the possibility to make DevOpsML independent from specific modeling technology. Indeed, for the sake of rapid proof of concept prototyping, the current DevOpsML implementation is built exclusively on EMF-compliant technologies. For example, its weaving mechanism is currently limited to EMF-based model artifacts. Finally, we will also investigate the availability of Eclipse-based technologies for software process modeling (e.g., Eclipse Process framework [18]).

5 RELATED WORK

As witnessed by recent endeavors by the MDE research community [6], cross-fertilization activities between MDE and DevOps can be divided in two categories, MDE for DevOps and DevOps for MDE, with the intent of a rapid uptake and adaptation of principles and practices from the former domain to the latter, and vice-versa. This work contributes to apply MDE for DevOps. In particular it focuses on software process and platform modeling for DevOps.

In [17], Dumas et al. introduces the concept of process-aware information system (PAIS) as *a software system that manages and* *executes operational processes involving people, applications, and/or information sources on the basis of process models.*. In this regard, DevOpsML can seen as a model-driven approach to create models of PAIS, emphasizing the importance of (modeling) languages and tools and their integration, and DevOpsML framework models as high-level architectural descriptions of model-driven, DevOps-wise PAIS.

In [14], a research roadmap and challenges to combine MDE and DevOps to bridge the gaps from development to operation and from operation to development phases are presented. Our framework tackles some of these challenges (integration of MDE techniques, integration of heterogeneous artifacts, selection of languages for Dev-to-Ops and Ops-to-Dev pipelines). Indeed, any MDE methodology that can be described as a set of modeling concerns, modeling languages, and tools can contribute (i) to populate libraries of reusable platform elements, (ii) be reused to create platform models and, (iii) integrated with process models in DevOpsML framework models.

In [7], Bordeleau et al. investigated the requirements of model-driven, DevOps-wise engineering platforms and they identified a set of requirement categories (general, description, and analysis and simulation requirements), including the need for considering process, product, and resource aspects [34], and proper modeling and tool integration mechanisms. We are currently evaluating DevOpsML against those requirements. A preliminary assessment is provided on the project repository [5]. It is worth noting that support for concrete requirement specifications can be expressed in DevOpsML by (i) directly as required concerns (see Figure 3) as part of DevOpsML platform configurations or (ii) specifying support for requirement specification by integrating languages and tools dedicated to requirement specifications. In both cases, requirements can be included in DevOpsML framework model.

The need for *process modeling* is recognized since years and encompasses any domain where there is the need for describing and executing complex organizational behaviors [9].

SPEM [29] is a standard process modeling language for descriptive process modeling. Different approaches use it as a baseline for more sophisticated approaches. In [35], Simmonds et al. introduce the *Software Process Lines (SPrL)* concept and a supporting model-driven framework, to deal with variability and evolution of software processes. They use SPEM [29] for modeling the software process. In DevOpsML, we are adopting a descriptive approach for process modeling and do not provide additional capabilities for process model management. We expect such a capability as externally provided by integrated tools and languages used in the DevOps domain [11].

In the context of DevOps, the adjective *continuous* is commonly used to refer to DevOps processes. In this paper, we used the acronym CSE introduced in [20] to refer to this category of engineering processes, whose shared concern is supporting a seamless integration of Dev and Ops phases. In particular, in [20], Garcia et al. investigated model-driven continuous delivery for teams. They show how typical MDE architecture for DevOps frameworks, with MDE artifacts (metalanguages, metamodels, models) and tools (e.g., model transformations for specific purposes [26]), can be realized to support CSE processes. In DevOpsML, we aim at populating libraries of languages and tools with non-MDE or MDE-ready platform elements (i.e., languages belonging to modeling spaces [16]).

Babar et al. [3] point out the need for configurable DevOps processes to cope with continuously evolving scenarios. They propose a Business Process Architecture (BPA) approach, based o variations points on BPMN models. DevOpsML does not choose a particular operational process modeling language and does not provide direct support to process modeling activities. For these reasons, it represents a candidate complementary for DevOpsML, deserving further investigation for potential future integration work.

Wurster et al. [39] perform a systematic review of the declarative deployment technologies and propose the *Essential Deployment Meta-Model* (EDMM) for comparison, selection, and migration of deployment technologies and to help users to select the one that best fits their scenario. Therefore, we will further investigate this work for populating DevOpsML tools and interface platform component library. Moreover, we will consider it during process and platform weaving phase as guidance for mapping deployment technologies to deployment activities.

Wettinger [38] presents a *gather'n'deliver* approach for collecting, describing, and integrating infrastructural software components and DevOps tooling (a.k.a. DevOpsware) as components within working DevOps platforms. An application environment metamodel and component taxonomies are provided for modeling DevOps processes and platform, together with a mechanism to generate DevOps platform skeletons from a repository of reusable components. Process and platform aspects of DevOps are then taken into account both in [38] and in DevOpsML. Moreover, the current DevOpsML prototype is not providing any generation mechanism. However, DevOpsML emphasizes the need for languages. We consider languages, like scripting languages typically used by DevOpsware tools or DSLs needed by engineers in support of arbitrary MDE methodologies, as first class architectural elements of any model-driven DevOps platform.

6 CONCLUSIONS

In this paper, we outlined the first prototype of DevOpsML, a model-driven framework for modeling DevOps processes and platforms and their weaving. We provided some preliminary guidelines and example of its use for documentation purposes. Furthermore, we outlined a research roadmap to guide our next research endeavors towards the combination of DevOps and MDE principles and practices with a focus on LCEPs.

ACKNOWLEDGMENTS

This project has received funding from the EU Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884.

REFERENCES

[1] João Paulo A. Almeida, Adrian Rutle, Manuel Wimmer, and Thomas Kühne. 2019. The MULTI Process Challenge. In *22nd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion, MODELS Companion 2019, Munich, Germany, September 15-20, 2019*, Loli Burgueño, Alexander Pretschner, Sebastian Voss, Michel Chaudron, Jörg Kienzle, Markus Völter, Sébastien Gérard, Mansooreh Zahedi, Erwan Bousse, Arend Rensink, Fiona Polack, Gregor Engels, and Gerti Kappel (Eds.). IEEE, 164–167. https://doi.org/10.1109/MODELS-C.2019.00027

[2] Atlassian. 2019. Gitflow Workflow. https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow/, last accessed on 28/08/20.

[3] Zia Babar, Alexei Lapouchnian, and Eric Yu. 2015. Modeling DevOps Deployment Choices Using Process Architecture Design Dimensions. In *The Practice of Enterprise Modeling*, Jolita Ralyté, Sergio España, and Óscar Pastor (Eds.). Springer International Publishing, Cham, 322–337.

[4] Jean Bézivin. 2006. Model Driven Engineering: An Emerging Technical Space. In *Generative and Transformational Techniques in Software Engineering: International Summer School, GTTSE 2005, Braga, Portugal, July 4-8, 2005. Revised Papers*, Ralf Lämmel, João Saraiva, and Joost Visser (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 36–64. https://doi.org/10.1007/11877028_2

[5] Linz BISE Institute, JKU. 2020. DevOpsML. https://github.com/lowcomote/devopsml/tree/1.2.2, last accessed on 28/08/20.

[6] Modeling Languages Blog. 2019. DevOps for models and modeling DevOps. https://modeling-languages.com/devops-modeling-workshop/, last accessed on 28/08/20.

[7] Francis Bordeleau, Jordi Cabot, Juergen Dingel, Bassem S. Rabil, and Patrick Renaud. 2020. Towards Modeling Framework for DevOps: Requirements Derived from Industry Use Case. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer (Eds.). Springer International Publishing, Cham, 139–151.

[8] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2017. *Model-Driven Software Engineering in Practice, Second Edition.* Morgan & Claypool Publishers. https://doi.org/10.2200/S00751ED2V01Y201701SWE004

[9] E. Breton and J. Bezivin. 2001. Process-centered model engineering. In *Proceedings Fifth IEEE International Enterprise Distributed Object Computing Conference.* IEEE, 179–182. https://doi.org/10.1109/EDOC.2001.950436

[10] Loli Burgueño, Federico Ciccozzi, Michalis Famelis, Gerti Kappel, Leen Lambers, Sébastien Mosser, Richard F. Paige, Alfonso Pierantonio, Arend Rensink, Rick Salay, Gabriele Taentzer, Antonio Vallecillo, and Manuel Wimmer. 2019. Contents for a Model-Based Software Engineering Body of Knowledge. *Software and Systems Modeling* 18, 6 (2019), 3193–3205. https://doi.org/10.1007/s10270-019-00746-9

[11] Necco Ceresani. 2016. The Periodic Table of DevOps Tools v.2 is Here. https://blog.xebialabs.com/2016/06/14/periodic-table-devops-tools-v-2/, last accessed on 28/08/20.

[12] Federico Ciccozzi, Matthias Tichy, Hans Vangheluwe, and Danny Weyns. 2019. Blended Modelling - What, Why and How. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C).* ACM/IEEE, 425–430. https://doi.org/10.1109/MODELS-C.2019.00068

[13] Benoît Combemale, Ralf Lämmel, and Eric Van Wyk. 2018. SLEBOK: The Software Language Engineering Body of Knowledge (Dagstuhl Seminar 17342). In *Dagstuhl Reports*, Vol. 7. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[14] Benoît Combemale and Manuel Wimmer. 2019. Towards a Model-Based DevOps for Cyber-Physical Systems. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment - Second International Workshop, DEVOPS 2019, Château de Villebrumier, France, May 6-8, 2019, Revised Selected Papers (Lecture Notes in Computer Science, Vol. 12055)*, Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer (Eds.). Springer, 84–94. https://doi.org/10.1007/978-3-030-39306-9_6

[15] Yingnong Dang, Qingwei Lin, and Peng Huang. 2019. AIOps: Real-world challenges and research innovations. *Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering: ICSE-Companion* (2019), 4–5. https://doi.org/10.1109/ICSE-Companion.2019.00023

[16] Dragan Djurić, Dragan Gašević, and Vladan Devedžić. 2006. The Tao of modeling spaces. *Journal of Object Technology* 5, 8 (2006), 125–147. https://doi.org/10.5381/jot.2006.5.8.a4

[17] Marlon Dumas, Wil M Van der Aalst, and Arthur H Ter Hofstede. 2005. *Process-aware information systems: bridging people and software through process technology.* John Wiley & Sons.

[18] Eclipse Fou. 2018. Eclipse Process Framework. https://www.eclipse.org/epf/, last accessed on 28/08/20.

[19] Eclipse Fou. 2019. Eclipse Modeling Framework. www.eclipse.org/modeling/emf/, last accessed on 28/08/20.

[20] Jokin Garcia and Jordi Cabot. 2019. Stepwise Adoption of Continuous Delivery in Model-Driven Engineering. In *Software Engineering Aspects of Continuous Development and New Paradigms of Software Production and Deployment*, Jean-Michel Bruel, Manuel Mazzara, and Bertrand Meyer (Eds.). Springer International Publishing, Cham, 19–32.

[21] Julián Alberto García-García, José Gonzalez Enríquez, and Francisco José Domínguez Mayo. 2019. Characterizing and evaluating the quality of software process modeling language: *Comparison of ten representative model-based languages.* *Comput. Stand. Interfaces* 63 (2019), 52–66. https://doi.org/10.1016/j.csi.2018.11.008

[22] Gartner. 2015. Gartner Says By 2016, DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations. https://tinyurl.com/y556a8moU, last accessed on 28/08/20.

[23] Ramtin Jabbari, Nauman bin Ali, Kai Petersen, and Binish Tanveer. 2016. What is DevOps? A Systematic Mapping Study on Definitions and Practices. In *Proceedings of the Scientific Workshop Proceedings of XP2016* (Edinburgh, Scotland, UK) *(XP '16 Workshops).* Association for Computing Machinery, New York, NY, USA, Article 12, 11 pages. https://doi.org/10.1145/2962695.2962707

[24] Dimitrios Kolovos, Louis Rose, Richard Paige, and Antonio Garcıa-Domınguez. 2010. The Epsilon book. *Structure* 178 (2010), 1–10.

[25] Leonardo Leite, Carla Rocha, Fabio Kon, Dejan Milojicic, and Paulo Meirelles. 2019. A Survey of DevOps Concepts and Challenges. *ACM Comput. Surv.* 52, 6, Article 127 (Nov. 2019), 35 pages. https://doi.org/10.1145/3359981

[26] Levi Lúcio, Moussa Amrani, Juergen Dingel, Leen Lambers, Rick Salay, Gehan M.K. K Selim, Eugene Syriani, and Manuel Wimmer. 2016. Model transformation intents and their properties. *Software and Systems Modeling* 15, 3 (2016), 647–684. https://doi.org/10.1007/s10270-014-0429-x

[27] Alexandra Mazak and Manuel Wimmer. 2016. Towards Liquid Models: An Evolutionary Modeling Approach. In *2016 IEEE 18th Conference on Business Informatics (CBI)*, Vol. 01. IEEE, 104–112. https://doi.org/10.1109/CBI.2016.20

[28] Håvard Myrbakken and Ricardo Colomo-Palacios. 2017. DevSecOps: a multivocal literature review. In *International Conference on Software Process Improvement and Capability Determination*, Vol. 770. Springer, Springer Verlag, 17–29. https://doi.org/10.1007/978-3-319-67383-7_2

[29] OMG. 2008. SPEM. https://www.omg.org/spec/SPEM/About-SPEM/, last accessed on 28/08/20.

[30] OMG. 2018. Semantics of a Foundational Subset for Executable UML Models. https://www.omg.org/spec/FUML/, last accessed on 28/08/20.

[31] George Pallis, Demetris Trihinas, Athanasios Tryfonos, and Marios Dikaiakos. 2018. DevOps as a Service: Pushing the Boundaries of Microservice Adoption. *IEEE Internet Computing* 22, 3 (2018), 65–71. https://doi.org/10.1109/MIC.2018.032501519

[32] Dewayne E. Perry and Alexander L. Wolf. 1992. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes* 17, 4 (1992), 40–52. https://doi.org/10.1145/141874.141884

[33] Apurvanand Sahay, Arsene Indamutsa, Davide Di Ruscio, and Alfonso Pierantonio. 2020. Supporting the understanding and comparison of low-code development platforms. In *Proceedings of the 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA).* IEEE.

[34] Miriam Schleipen and Rainer Drath. 2009. Three-view-concept for modeling process or manufacturing plants with AutomationML. In *Proceedings of ETFA 2009 - 2009 IEEE Conference on Emerging Technologies and Factory Automation.* IEEE, 1–4. https://doi.org/10.1109/ETFA.2009.5347260

[35] J. Simmonds, D. Perovich, M. C. Bastarrica, and L. Silvestre. 2015. A megamodel for Software Process Line modeling and evolution. In *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS).* IEEE, 406–415. https://doi.org/10.1109/MODELS.2015.7338272

[36] Massimo Tisi, Jean-Marie Mottu, Dimitrios S Kolovos, Juan de Lara, Esther Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. 2019. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *{STAF} (Co-Located Events) ({CEUR} Workshop Proceedings, Vol. 2405).* CEUR-WS.org, 73–78.

[37] Klaas van den Berg, J.M. Conejero, and R Chitchyan. 2005. *AOSD Ontology 1.0: Public Ontology of Aspect-Orientation.* Number AOSD-E in AOSD-Europe-UT-01. AOSD Europe. AOSD-Europe-UT-01.

[38] Johannes Wettinger. 2017. *Gathering solutions and providing APIs for their orchestration to implement continuous software delivery.* Ph.D. Dissertation. University of Stuttgart, Germany. https://nbn-resolving.org/urn:nbn:de:bsz:93-opus-ds-91108

[39] Michael Wurster, Uwe Breitenbücher, Michael Falkenthal, Christoph Krieger, Frank Leymann, Karoline Saatkamp, and Jacopo Soldani. 2020. The essential deployment metamodel: a systematic review of deployment automation technologies. *SICS Softw.-Intensive Cyber Phys. Syst.* 35, 1 (2020), 63–75. https://doi.org/10.1007/s00450-019-00412-x