Promote-pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines

Jacob Krüger
University of Toronto &
Otto-von-Guericke University
Canada & Germany

Wardah Mahmood
Chalmers | University of Gothenburg
Sweden

Thorsten Berger
Chalmers | University of Gothenburg
Sweden

ABSTRACT

Process models for software product-line engineering focus on proactive adoption scenarios—that is, building product-line platforms from scratch. They comprise the two phases domain engineering (building a product-line platform) and application engineering (building individual variants), each of which defines various development activities. Established more than two decades ago, these process models are still the de-facto standard for steering the engineering of platforms and variants. However, observations from industrial and open-source practice indicate that the separation between domain and application engineering, with their respective activities, does not fully reflect reality. For instance, organizations rarely build platforms from scratch, but start with developing individual variants that are re-engineered into a platform when the need arises. Organizations also appear to evolve platforms by evolving individual variants, and they use contemporary development activities aligned with technical advances. Recognizing this discrepancy, we present an updated process model for engineering software product lines. We employ a method for constructing process theories, building on recent literature as well as our experiences with industrial partners to identify development activities and the orders in which these are performed. Based on these activities, we synthesize and discuss the new process model, called *promote-pl*. Also, we explain its relation to modern software-engineering practices, such as continuous integration, model-driven engineering, or simulation testing. We hope that our work offers contemporary guidance for product-line engineers developing and evolving platforms, and inspires researchers to build novel methods and tools aligned with current practice.

CCS CONCEPTS

• **Software and its engineering** → *Software evolution*; *Software product lines*; *Software development process management*.

KEYWORDS

Software reuse, process model, round-trip engineering

ACM Reference Format:
Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines. In *24th ACM International Systems and Software Product Line Conference (SPLC '20), October 19–23, 2020, MONTREAL, QC, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3382025.3414970

1 INTRODUCTION

Software product-line engineering provides methods and tools for building variant-rich systems. It allows to systematically reuse software features (i.e., user-visible functionalities of a system) by establishing an integrated software platform [3, 63, 82]. To build a platform, developers employ a range of implementation techniques called variability mechanisms [3, 32] to define variation points. Individual variants can then be derived through configuring—enabling or disabling features. Typical variability mechanisms comprise preprocessors (e.g., the C preprocessor), configurable build systems, configurator and variant-derivation tools [14, 49, 69], as well as model-based representations of features and their constraints, called variability models [11, 21, 76, 92]. Especially the latter are core to manage features and to guide the derivation of individual variants.

While the underlying ideas and mechanisms employed remain similar, their implementation and usage have evolved considerably over the last decades, enabling organizations to rely on more advanced automation. Examples of these advancements are novel analysis techniques for feature models, code, and test assets [7, 74, 100, 102], or the adoption of continuous integration [71, 97]. Many of these techniques have implications on the processes with which variant-rich systems are engineered. Unfortunately, the process models for product-line engineering have not been updated accordingly. Consider one of the most common process models for product-line engineering [82], as shown in Fig. 2. This model strictly distinguishes between domain engineering (i.e., developing the platform) and application engineering (i.e., developing variants), defining five and four activities, respectively. This process model should be updated to reflect, for example, the less strict separation of domain and application engineering in practice, the evolution

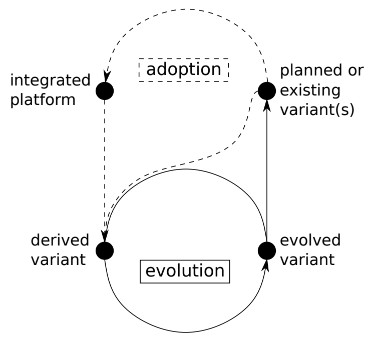

Figure 1: High-level representation of promote-pl.

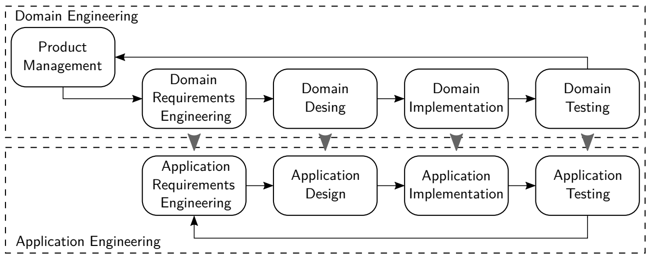

Figure 2: A typical product-line process model [82].

of product lines via their variants, and different adoption strategies [48] (we detail these examples in Sec. 2). In short, we believe that the core limitations of existing process models are the strict separation of domain and application engineering, and the focus on the proactive adoption strategy, which, as we will show, do not reflect industrial and open-source practice anymore.

We present *promote-pl* (PROcess MOdel for round-Trip Engineering of Product Lines), an updated process model for product-line engineering that we synthesized from recent literature and collaborations with industry. For this purpose, we adapted methods for deriving process theories [85, 95] to elicit empirical data and synthesize promote-pl (high-level representation in Fig. 1, explained in Sec. 4.2). We further discuss the adaptations we implemented in promote-pl and analyze its relations to contemporary software-engineering practices to define research opportunities. In detail, we contribute:

- A systematically elicited process model for product-line engineering that reflects recent practices, called *promote-pl*.
- A discussion of the adaptations we implemented and their implications for practice as well as research.
- An analysis of relations between promote-pl and software-engineering practices.

With these contributions, we intend to provide a more realistic and updated process model for product-line engineering that can provide a better understanding of how organizations engineer software platforms. Especially for researchers, promote-pl highlights the differences between historically defined process models and industrial practices, helping them to identify research opportunities.

2 MOTIVATION AND OBJECTIVES

In Fig. 2, we illustrate the structure of a typical product-line process model [82]. We can see that the domain engineering encompasses a special activity for product management as well as activities for requirements engineering, design, implementation, and testing of the platform. Moreover, there is a loop between these activities, indicating evolution of the platform. Strictly separated and always building on the defined platform is the application engineering with the respective four activities for deriving a variant. This process model is a high-level abstraction (i.e., Pohl et al. [82] describe the activities in more detail, as well as the need to tailor the model to concrete systems) that is similar to other established process models, most notably those of Northrop [79], Kang et al. [44], Czarnecki [20], and Apel et al. [3]. However, even though some of these process models encompass minor differences, they appear to not be in line with contemporary practices [13], they largely disregard recent trends of blurred boundaries between software-engineering phases (e.g., continuous software engineering [29]), and they focus on the proactive adoption strategy (explained shortly). For example, Pohl et al. suggest to conduct the "Commonality analysis first," supporting the proactive adoption—that is, first establishing a platform before individual variants are derived. In contrast, our and others' experiences with practitioners [5, 10, 11] show a dominance of reactive and extractive adoption strategies, where organizations start with one or multiple variants first, before eventually establishing a platform. This means that, for instance, activities such as variability analysis are performed later [76] than prescribed by the traditional process models. Furthermore, platforms are typically evolved via variants—customers request additional features, which are first introduced into variants and later integrated (e.g., back-propagated) into the platform [60]. In this paper, we describe promote-pl as an updated process model for product-line engineering to reflect contemporary practices as well as adoption and evolution processes that are predominant in current practice [5, 53, 60, 84].

2.1 Example Limitations of Process Models

We exemplify (mingled) limitations of existing process models, quoting insights from the company Danfoss with its long-living and well-documented [30, 41–43] product line of frequency converters in the power electronics domain. The experiences are largely in line with our own experiences from studying industrial practice [9, 10, 12, 13, 33, 53, 54, 57, 60, 61, 64, 76, 93].

Separation of Domain and Application Engineering. We experienced that most organizations and developers do not strictly separate (or even distinguish) domain and application engineering. Instead, there is constant interaction between both. For example, features are often implemented in a variant and later integrated into the platform (see second example), for reactive and extractive adoption the platform is even defined based on existing variants (see third example), and processes iterate between platform and variant (e.g., during testing). Similarly, Danfoss experienced [30]:

> "[...] there was no strict separation between domain and application engineering in the product projects [...]"

In their case, the main idea was to limit the number of changes required to adopt processes and tool chains; facilitating an extractive adoption. So, we argue that we need a new process model that integrates interactions between domain and application engineering.

Evolution of the Product Line. Existing process models define that new requirements are propagated to the domain engineering, features are implemented on the platform, and the variant is derived afterwards. This is the ideal scenario, but most organizations and open-source projects use the well-known concept of feature forks [57, 60, 98] to implement new variants or platform features. By re-integrating these forks, the platform is evolved—but this is driven by developing and merging complete or partial variants. Obviously, missing to re-integrate the variants results in clone & own development instead of product-line engineering. Still, Danfoss experienced that feature forks allow that [30]:

> "[...] projects could keep their independence by introducing product-specific artifacts as new features. Later on, when a change was assessed, there would be a decision on whether the change should be applied to other products and thus should be integrated into the core assets."

The stated independence and also fast delivery are benefits of feature forks, and they align with continuous integration. Due to their practical importance, different evolution scenarios should be added to the process model.

Adoption Strategies. Even though, extractive and reactive adoption strategies are more common in practice [11], existing process models focus on the proactive adoption in which a product line is planned from scratch. However, due to the economic investments and risks [17, 50, 55, 60, 91], most organizations start with clone & own and only later migrate towards a product line. Consequently, they have a variety of existing variants from which they can, for example, recover architectures, reuse code, or analyze domain documentation to design the platform. This results in adopted, new, and re-ordered activities, depending on the adoption strategy. For instance, Danfoss [30] employed an extractive adoption, during which the organization migrated 80 % of the code and introduced continuous integration within the first five years, but:

> "Introducing pure::variants and establishing feature models for both code and parameters, and finally including the requirements, would take another two years."

As we can see in Fig. 2, this conflicts existing process models in which a feature model is defined before the implementation (i.e., in the domain design). So, a general process model for product-line engineering should also incorporate different adoption strategies.

2.2 Research Objectives

Our overarching goal was to derive an updated, practice-oriented process model for product-line engineering. This model should help practitioners as well as researchers in understanding current practices, fostering the adoption, improvement, and future research of product lines. To achieve this, we defined three research objectives:

- RO$_1$ Elicit empirical data about contemporary product-line engineering processes and activities employed in practice.
- RO$_2$ Synthesize a common process model that puts the identified activities into a reasonable order.
- RO$_3$ Discuss the adaptations in the process model and the impact of contemporary software-engineering practices.

According to these objectives, we defined an empirical methodology to elicit data (RO$_1$) and to construct the process model, promote-pl (RO$_2$). Promote-pl itself (RO$_2$) and our discussion of adaptations and practices (RO$_3$) represent the resulting contributions.

3 METHODOLOGY

Using a process model, we can describe *how* something happens in an actual, real-world process [19]. In contrast, development methodologies describe an assumed "best practice" of doing something, while a process theory is a universal description of a process [85]. We remark that researchers heavily debate about what a process theory constitutes in detail, and some definitions are close or even identical to a process model [85, 95]. However, following the distinction of Ralph [85], we define a process model, since we focus on constructing a process from empirical evidence, neither claiming that it represents best practices nor that it can explain all existing processes for product-line engineering in their entirety. As we can only cover the product-line engineering activities that we could identify, these two properties can, arguably, not be fulfilled; considering, for example, the numerous tools, implementation techniques, or testing strategies that exist. Moreover, future advances in research may require changes in promote-pl.

We are not aware of a specific guideline for constructing process models. Instead, we adapted recommendations for deriving process theories [85, 95]. As a result, we relied on three information sources:

- First, each author suggested publications based on their knowledge of the literature, without relying on a systematic search (cf. Sec. 3.1). This design resembles integrative reviews [96], which are helpful to critically reflect, synthesize, and re-conceptualize theoretical models for mature research areas—which was our research goal.
- Second, we extended the suggested publications based on a systematic literature review [45], searching manually in the last five instances of relevant venues (cf. Sec. 3.2). Our goal was to more systematically and extensively cover the most recent developments in product-line engineering to understand, incorporate, and discuss current practices.
- Finally, we relied on our own experiences (also adding the corresponding publications) of collaborating with industrial partners that employ product-line engineering (cf. Sec. 3.3). We used our experiences to structure our data, order activities, and discuss how practices are aligned with promote-pl.

By using these information sources, we base promote-pl in empirical evidence to strengthen its validity. In the following, we describe each information source in more detail, our strategy to elicit data from the publications identified (cf. Sec. 3.4), and how we synthesized the data to construct promote-pl (cf. Sec. 3.5).

3.1 Knowledge-Based Literature Selection

We used our knowledge of the literature and particularly from recently conducted (semi-)systematic literature reviews [13, 60, 76] to select publications. For this purpose, each author suggested publications that they considered relevant, based on a publication's topicality and relevancy for our research goal. We discussed each suggestion based on the following **inclusion criteria**, and only incorporated a publication if we achieved mutual agreement:

- IC$_1$ The publication is written in English.
- IC$_2$ The publication describes activities of product-line engineering, suggesting at least one partial order (i.e., a minimum of two activities in a sequence of execution).
- IC$_3$ The publication reports activities based on recent (i.e., five years) experiences (e.g., case studies, interviews) or synthesizes them from such experiences (e.g., literature reviews).

We performed an initial selection to scope our research, but also added publications later in our analysis.

Results. In the beginning, we selected five publications that describe well-known process models for product-line engineering (cf. Sec. 2) as baseline for our work—marked as BL in Tbl. 1. We included these publications to have a foundation that we could extend and refine to construct promote-pl. Note that we included the publication of Northrop [79], due to the reported process model being well established, even though it does not fulfill IC$_2$ (no partial orders). Furthermore, we agreed to add 12 additional suggestions (marked with ER in Tbl. 1) that cover up-to-date experiences (i.e., IC$_3$) — including publications with our experiences (marked with *).

3.2 Systematic Literature Selection

To define a more systematic foundation for the process model, we decided to perform the search and selection phase of a systematic literature review [45]. So, we did not only rely on our own knowledge, but extended our information sources using a replicable process.

Search. We conducted a manual search among five conferences (SPLC, VaMoS, ICSE, ESEC/FSE, ASE) and seven journals (TSE, EMSE, TOSEM, JSS, IST, IEEE Software, SPE); aiming to avoid the problems of automated searches [40, 56, 94]. For the conferences, we covered their last five editions of research and industry tracks, including the 2015 to 2019 (and additionally 2020 for VaMoS) editions for each. For the journals, we considered the years from 2016 to 2020, including online-first publications. We selected these time spans to consider current product-line practices for promote-pl.

To conduct the search, we used DBLP as of April 7th 2020—except for online-first publications, for which we relied on the journals' websites as of that same date. We selected major software engineering venues that employ peer-reviews and publish product-line research, ensuring the quality of included publications. While we certainly miss some publications that describe product-line engineering processes, we argue that this selection provides a reasonable overview of recent publications to understand what adaptations are required to design a contemporary process model [96].

Inclusion Criteria. To select relevant publications, we employed the same inclusion criteria as for the knowledge-based selection. Further, we essentially added two more inclusion criteria:

- IC$_4$ The publication has been published at the research or industry track of a peer-reviewed venue.
- IC$_5$ The publication does not only propose a process (e.g., new testing methods), but this process is actually used in practice.

Using these criteria, we ensured that the selected publications actually cover real-world processes and not only proposals, for example, for incorporating a new research tool.

Results. With the manual search, we identified 16 new publications, which we mark with SR in Tbl. 1. Note that we do not account for publications we already identified in the previous search in this set. In the end, we selected 33 publications for constructing promote-pl.

3.3 Industrial Collaborations

We regularly collaborate with different industrial partners that employ product-line engineering. For instance, we worked with 12 medium- to large-sized organizations to assess their state of adopting variability management [13], interviewed experts to understand feature-modeling practices [76], and collaborated with large organizations, such as Axis [60], Saab [64], or ABB, to improve our understanding of product-line practices. We used our gained knowledge, resulting publications, and ongoing discussions, to reason about the data we elicited from the literature. Particularly, we resolved unclear partial orders to construct promote-pl (Sec. 3.5) and based the discussion of software-engineering practices on this knowledge.

3.4 Data Extraction

For every publication, we extracted standard bibliographic data, namely authors, title, as well as publication venue and year. To construct promote-pl, we further extracted all product-line engineering activities (i.e., we did not consider "standard" activities, such as requirements elicitation) that have been mentioned in their specific wording. If these activities were in a partial order, we also extracted that order. Moreover, we extracted the scope in which these activities have been applied, for example, extractive adoption or platform-based evolution. Finally, if we identified a specific software-engineering practice to be used, we also documented this. We used a table to document and manage this data—with Tbl. 1 providing a summary of that table.

3.5 Process Construction

To construct promote-pl, we executed the following steps:

1. We collaboratively analyzed the process models presented in the five baseline publications (marked with BL in Tbl. 1). So, we obtained an initial understanding of the existing process models, how to unify terminologies, and a first set of partial orders. However, the most important outcome was a mutual agreement on how to elicit and document partial orders.
2. Every author suggested relevant publications, and the first author conducted the manual search.
3. The first author read each publication, decided whether it fulfilled the inclusion criteria, and extracted the data described in Sec. 3.4 if this was the case. To ensure that we did not miss important publications or activities, the other authors verified distinct subsets of all publications.
4. We created a list of unique activity names (150+), which the first author used to resolve synonyms, specify terms, and abstract common activities. For example, we changed all occurrences of "product" or "system" to "variant," and specified "analyze requirements" according to its context (i.e., platform, asset, or variant). The employed changes were verified and agreed upon by the other authors. We remark that we were careful and aimed not to overly abstract activities (e.g., we kept "build" as a detailed activity of "derive variant"), which is why we report 99 distinct activities in Tbl. 1.
5. We compared the different partial orders and activities based on their scope and similarities. As a result, we defined partitions of the process model (e.g., adoption and evolution).
6. We constructed the process model by merging partial orders. To this end, the first author used re-appearing activities and similarities in the orders, structuring these according to the identified partitions. Then, we removed redundancies as far as possible to derive a unified process model.
7. To verify and agree on promote-pl, the third author interviewed the first author. During this interview, the first author explained promote-pl, design decisions, potential alternative representations, and based on what data each model element was incorporated. We agreed to employ smaller changes in promote-pl to improve its comprehensibility and resolve unclear orders of activities.

By using this methodology, we aimed to improve the validity of promote-pl, allowing other researchers to verify and replicate it.

4 THE PROCESS MODEL PROMOTE-PL

We describe the partial orders of activities we identified from the literature, followed by the structure and details of promote-pl.

Table 1: Overview of the 33 publications we analyzed and the activities described (based on our unified terminology).

	Ref.	**Venue**	**Scope**	**Activities in their Partial Orders (≺): • – Separator; & – Parallelism; | – Alternatives; [. . .] – Sub-activities**
BL	[44]	IEEE SW'02	Pro. Ado.	Scope & Budget Platform ≺ Analyze Platform Requirements & Model Variability ≺ Design Architecture ≺ Design System Model ≺ Refine Architecture ≺ Design Assets • Analyze Variant Requirements & Select Features ≺ Design & Adapt Architecture ≺ Adapt Assets & Build Variant
BL	[79]	IEEE SW'02	Pro. Ado.	Develop Assets • Engineer Variant • Manage Platform • Design Architecture • Evaluate Architecture • Analyze Platform Requirements • Integrate Assets • Identify Assets • Test • Configure • Scope Platform • Train Developers • Budget Platform
BL	[20]	UPP'04	Pro. Ado.	Analyze Domain ≺ Design Architecture ≺ Implement Platform • Analyze Variant Requirements ≺ Derive Variant
BL	[82]	Book'05	Pro. Ado.	Scope Platform ≺ Analyze Domain [Analyze Commonalities ≺ Analyze Variability ≺ Model Variability] ≺ Design Architecture ≺ Implement Platform ≺ Test Platform • Analyze Variant Requirements ≺ Design Variant ≺ Derive Variant [Configure ≺ Implement Specifics ≺ Build Variant] ≺ Test Variant
BL	[3]	Book'13	Pro. Ado.	Analyze Domain [Scope Platform ≺ Model Variability] ≺ Implement Platform ≺ Analyze Variant Requirements ≺ Derive Variant
ER	[88]	STTT'15	Ext. Ado.	Analyze Commonality & Variability [Compare Requirements ≺ Diff Variants ≺ Model Variability] ≺ Design Architecture [Extract Architecture ≺ Evaluate Architecture ≺ Refine Architecture & Variability Model] ≺ Develop Assets • Merge Variants ≺ Refactor ≺ Add Variation Points [Diff Variants ≺ Refactor] ≺ Model Variability ≺ Derive Variant • Analyze Commonality & Variability [Model Variability ≺ Compare Requirements & Tests & Diff Variants ≺ Refine Variability Model] ≺ Extract Platform
ER	[30]	SPLC'16	Ext. Ado.; Vb. Evo.	Diff Variants ≺ Analyze Variability ≺ Model Variability ≺ Add Variation Points ≺ Adopt Tooling ≺ Compare Requirements ≺ Map Artifacts • Develop Assets [Propose Asset ≺ Analyze Asset Requirements ≺ Design Asset ≺ Implement Asset ≺ Test Asset] • Release Platform [Plan Release ≺ Produce Release Candidate ≺ Test Platform] • Release Variant [Scope Variant ≺ Derive Variant ≺ Test Variant]
ER	[5]	ESE'17	Ext. Ado.	Analyze Commonality & Variability [Locate Features] ≺ Model Variability ≺ Re-Engineer Artifacts
ER*	[58]	SPLC'17	Ext. Ado.	Diff Variants ≺ Locate Features ≺ Model Variability ≺ Map Artifacts
ER*	[61]	SPLC'18	Ext. Ado.	Model Variability ≺ Adopt Tooling • Domain Analysis • Implement Platform • Analyze Variant Requirements • Derive Variant • Configure
ER	[68]	SPLC'18	Ext. Ado.	Train Developers ≺ Analyze Domain ≺ Model Variability ≺ Implement Assets [Analyze Documentation | Diff Variants ≺ Refactor]
ER*	[54]	Chapter'19	Ext. Ado.	Analyze Variability ≺ Locate Features ≺ Map Artifacts
ER*	[76]	ESEC/FSE'19	Ado.; Evo.	Plan Variability Modeling ≺ Train Developers ≺ Model Variability ≺ Assure Quality [Evaluate Model • Test Model]
ER*	[57]	JSS'19	Vb. Evo.	Propose Asset ≺ Analyze Asset Requirements ≺ Assign Developers ≺ Fork Platform ≺ Implement Asset ≺ Create Pull-Request ≺ Review Asset ≺ Merge into Test Environment ≺ Test Asset ≺ Merge into Platform ≺ Release Platform
ER*	[100]	SPLC'19	Ext. Ado.; Vb. Evo.	Adapt Variant ≺ Propagate Adaptations • Analyze Domain ≺ Analyze Variability ≺ Locate Features • Extract Platform • Model Variability • Extract Architecture • Refactor • Test Platform • Test Variant
ER*	[60]	ESEC/FSE'20	Vb. Evol.	Scope Variant ≺ Design Variant ≺ Derive Variant ≺ Adapt Variant ≺ Assure Quality
ER*	[53]	VaMoS'20	Ext. Ado.	Train Developers ≺ Analyze Domain ≺ Prepare Variants [Remove Unused Code ≺ Translate Comments ≺ Analyze Commonality ≺ Diff Variants] ≺ Analyze Variability ≺ Extract Architecture ≺ Locate Features ≺ Model Variability ≺ Extract Platform ≺ Assure Quality
SR	[108]	SPLC'15	Vb. Evo.	Scope Variant [Analyze Variant Requirements • Design Variant • Configure] ≺ Budget Variant ≺ Design & Implement Variant [Analyze Variant Requirements ≺ Design & Evaluate Variant ≺ Implement & Adapt Variant ≺ — | Propagate Adaptations] ≺ Configure & Test Variant
SR	[105]	VaMoS'15	Ext. Ado.	Analyze Variability [Diff Variants & Identify Fork Points ≺ Classify Adaptations ≺ Merge Bug Fixes | [Name Assets ≺ Merge Assets into Hierarchy]] ≺ Add Variation Points ≺ Model Variability ≺ Locate Features ≺ Extract Platform ≺ Configure
SR	[46]	ESE'16	Ado.	Analyze Domain [Gather Information Sources ≺ Define Reuse Criteria ≺ Collect Information ≺ Analyze & Model Variability ≺ Extract Architectures ≺ Evaluate Results] ≺ Budget Platform
SR	[75]	SPLC'16	Pro. Ado.	Engineer Platform [Analyze Platform Requirements ≺ Design Architecture & Implement Platform ≺ Implement Assets] ≺ Derive Variants • Manage Platform
SR	[39]	SPLC'16	Pro./Ext. Ado.	Scope Platform ≺ Engineer Platform [Design System Model ≺ Design Architecture & Implement Platform ≺ Model Variability] ≺ Derive Variant [Design Variant [Design Variant Model ≺ Scope Variant ≺ Select Features] ≺ Evaluate Design [Evaluate Design Logic ≺ Configure] ≺ Design Variant ≺ Implement Variant] ≺ Test Variant
SR	[16]	VaMoS'16	Pb. Evo.	Analyze Variant Requirements ≺ Define Build Rules ≺ Configure & Derive Variant ≺ Test Variant
SR	[103]	JSS'17	Pro. Ado.	Model Variability ≺ Design System Model ≺ Derive Variant
SR	[34]	SPLC'17	Vb. Evo.	Fork Platform ≺ Test Platform ≺ Merge into Platform
SR	[107]	SPLC'17	Pro. Ado.	Design Architecture ≺ Add Variation Points ≺ Model Variability ≺ Configure ≺ Derive Variant
SR	[36]	SPLC'17	Vb. Evo.	Derive Variant [Scope Variant ≺ Plan Variant [Define Variant Backlog ≺ Estimate Efforts ≺ Plan Development] ≺ Build Variant [Create Backlog ≺ Time-Box Control]] • Manage Platform [Scope & Budget Platform]
SR	[18]	SPLC'17	Pro. Ado.	Analyze Platform Requirements [Analyze Domain ≺ Scope Platform ≺ Model Variability] ≺ Design Architecture ≺ Evaluate Architecture & Map Artifacts ≺ Derive Variant
SR	[83]	ICSE-SEIP'18	Ado.; Pro. Evo.	Analyze Platform Requirements ≺ Analyze Commonality & Variability ≺ Design Architecture ≺ Implement Platform • Analyze Variant Requirements ≺ Scope Variant [Identify Assets & Define New Assets] ≺ Implement Assets ≺ Integrate Assets ≺ Configure ≺ Test Variant • Map Artifacts • Model Variability • Unify Variability
SR	[35]	SPLC'18	Vb. Evo.	Define Variant Backlog ≺ Implement Variant [Analyze Variant Requirements ≺ Implement Assets ≺ Test Variant] ≺ Add Variation Points [Design Variation Points ≺ Refactor ≺ Test Platform]
SR	[90]	TSE'18	Ado./Evo.	Add Variation Points ≺ Adopt Tooling • Manage Knowledge • Resolve Configuration Failures • Assure Quality
SR	[67]	SPE'19	Ext. Ado.	Plan Development [Assign Developers ≺ Assign Roles ≺ Analyze Documentation] ≺ Assemble Process [Select Techniques ≺ Adopt Tooling ≺ Assign Tasks] ≺ Extract Platform [Execute Assembled Process ≺ Document Assets ≺ Document Process]
SR	[38]	SPLC'19	Pro. Ado.	Analyze Domain [Specify Properties ≺ Model Variability • Analyze Variant Requirements [Configure ≺ Optimization]] • Derive Variant [Configure ≺ Integrate Assets ≺ Test Variant] • Implement Platform

BL: BaseLine; ER: Expert Review; SR: Systematic Review
Ext.: Extractive; Pro.: Proactive; Ado.: Adoption; Pb.: Platform-based; Vb.: Variant-based; Evo.: Evolution

4.1 Contemporary PLE Practices (RO$_1$)

In Tbl. 1, we provide an overview of all 33 publications we considered. We can see that the publications we identified based on suggestions and the manual search cover mostly the extractive adoption strategy and evolution, which have become major topics in product-line engineering research [5, 8, 52, 65, 77, 100]. Moreover, the publications have been published in various venues, not surprisingly mostly at the flagship conference for software product-line engineering SPLC. We argue that this selection provides a broad and contemporary overview of practice, serving as a suitable dataset for adapting the baseline process models. However, we also identified interesting properties of the dataset that were important to consider while constructing promote-pl.

Activities. We unified the terminologies used in the selected publications, and abstracted activities to compare their orders. Still, we kept 99 unique activities, far too many to integrate into promote-pl. There are two reasons for this many activities. First, the publications vary heavily in the level of detail in which they report activities. For example, some simply state "derive product," while others detail single steps of this activity (e.g., "build"). Second, the publications cover various software-engineering methods (e.g., agile, model-driven), domains (e.g., power plants, web services), implementation techniques (e.g., C preprocessor, runtime variability), tools (e.g., fully automated derivation process, build system), and development phases (e.g., business analysis, variant derivation). The varying levels of details and the high diversity mean that it is not possible to unify all terms and activities. We addressed this issue by focusing on re-appearing activities in similar orders.

Partial Orders. As we can see in Tbl. 1, we obtained a total of 42 partial orders (without counting sub-orders or alternatives). Interestingly, due to the variations in the activities, there is not a single order that is identical to another order. Still, within a specific scope (e.g., extractive adoption), they share similarities in terms of activities and their orders—while they are quite different between scopes. This indicates again that we require an updated process model for product-line engineering.

Besides the high diversity of activities, one particular reason for the missing overlap seems to be ambiguity of what actions a specific activity comprises. For instance, "analyze domain," "scope platform," and "analyze commonality/variability" are often used together within partial orders. However, their exact orders vary, and sometimes one of these activities is a sub-activity of another. This indicates that it may not be well-understood what activities comprise what concrete actions, for instance, because different process models vary in their definitions. To tackle this problem, we read descriptions in the papers and relied particularly on the descriptions of Pohl et al. [82] to reason about design decisions.

4.2 Process Model Elements (RO$_2$)

We display the high-level abstraction of promote-pl in Fig. 1. The **adoption** includes starting from existing (extractive) or planned (proactive) variants that are integrated into a platform. Alternatively, a planned or existing variant can represent the derived variant that is extended later on (reactive). During the **evolution**, derived variants are evolved to include new features. Such variants can be evolved individually (clone & own) or integrated into the platform by merging features or variants (returning to **adoption**).

We show the detailed representation of promote-pl in Fig. 3, using a customized representation that builds on UML activity diagrams [80] to ease comprehensibility. The representation comprises nine different elements (summarized in the bottom left corner):

1) *Start Nodes* have essentially the same meaning as in UML, but we allow to start only at one; whereas UML would require to initiate the workflow at all start nodes simultaneously.

2–3) *Activities* and *Activity Edges* have exactly the same meanings and representations as in UML.

4) *Concurrent Activities* are similar to fork and join nodes in UML, indicating that the activities connected by the arrows are (or can be) performed at the same time

5) *Decision Nodes* have the same meaning as in UML, and we explicitly allow that they may have only one outgoing edge; representing an optional workflow.

6) We use *Situational Alternative* to easily represent two scenarios: First, to display that variant development also reflects parts of the reactive adoption strategy. Second, to show that one workflow occurs only if artifacts of variants are extracted (i.e., extractive adoption, evolution via variant integration).

7-9) We abstractly indicate the position of six processes and their workflows in promote-pl, distinguishing three different types. First, *Adoption Processes* (↓) are the proactive and extractive adoption strategies (as we will explain, reactive adoption represents an evolution process). Second, *Evolution Processes* (↻, ↺) are re-appearing workflows used to extend a product line—usually incorporating forward- and re-engineering activities (i.e., round-trip engineering). Third, the *Management Process* (↻) represents seven activities that are concerned with enabling and managing other processes, which is why they are performed constantly and in parallel.

We remark that we omit end nodes, since promote-pl reflects adoption and evolution in a round-trip engineering style. So, the end of all processes would mean that the product line is discontinued.

4.3 Promote-pl and Adaptations (RO$_2$, RO$_3$)

A Different Decomposition. As explained in Sec. 2, organizations appear to decreasingly separate and distinguish domain and application engineering, which is supported by our elicited data and experiences. For organizations, it is more important to understand how to adopt a platform and engineer variants, instead of considering the two phases in isolation (e.g., Danfoss reports platform extraction without fully modeling variability first [30]). Moreover, organizations have mixed teams that employ domain and application engineering in parallel. For example, in some organizations, the same team implements a new variant and refactors it into platform assets, whereas the platform team only tests and quality-assures assets. We reflected this primary concern of interest in promote-pl, moving from domain and application engineering towards the *Adoption* and *Evolution* of a product line. This is a major difference compared to existing process models, and we explain the resulting overarching processes of promote-pl in the following.

Product-Line Management. Some baseline process models comprise activities for managing a product line, often integrated into the

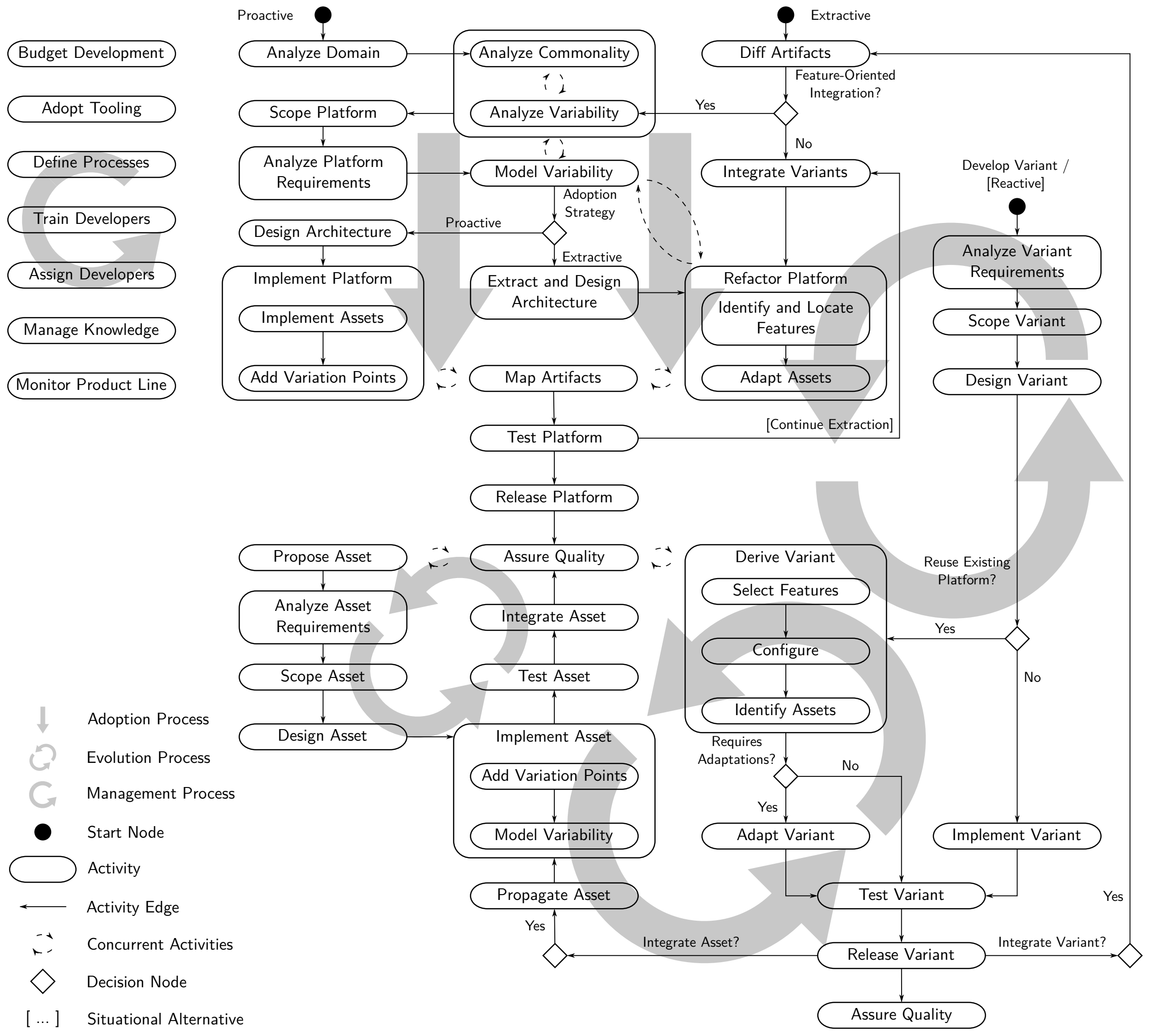

Figure 3: The detailed representation of promote-pl.

domain engineering, but also as a separate phase. Our empirical data suggests that the *management process* (↻) comprises a challenging and practically important set of activities, enabling organizations to plan and apply product-line engineering successfully. We found that all management activities should run in parallel—to each other and all development activities, which was also suggested before [79].

In particular, we found that seven activities are mentioned as important, for instance, budgeting development activities, adopting tooling as well as processes, and training developers, most of which are mingled and require monitoring of development activities for steering. Interestingly, such management activities have gained less interest in the research community compared to development activities [84]. For instance, budgeting may be supported by cost models, and several of such models have been proposed for product-line engineering. Unfortunately, existing cost models are often limited (e.g., considering their scopes and foundations in empirical data [1, 50, 53]), and only few experience reports provide guides on how to employ them in practice [46, 78].

Product-Line Aoption. For adopting a product line, we distinguish between the three strategies defined by Krueger [48].

First, the *proactive development process* (left ↓) is identical to the domain engineering of the baseline process models, comprising only minor clarifications. At the beginning, an organization analyzes its domain, comparing its commonalities (which others recommend to start with to identify reuse potential [82]) and variability. Based on the results, the platform is scoped and requirements are derived, which allows to construct a variability model. As we display in Fig. 3, variability modeling can be performed in parallel to analyzing commonality and variability, as both may affect each other (e.g., refining the variability model). Considering Fig. 2, this represents “domain requirements engineering” and “domain design” in the same order, but in a more flexible process. Afterwards, the platform architecture is designed and the platform with its assets as well as variation points is implemented (i.e., “domain implementation” in Fig. 2). We make two activities more explicit here that have been mentioned multiple times, and thus seem important to include: adding variation points and mapping artifacts (e.g., assets, documentation, variability model) to ensure traceability. Finally, the resulting product-line platform must be tested, released, and quality assured, again fully in line with baseline process models. Overall, this proactive adoption process is close to the domain engineering described by Pohl et al. [82]—except for separating management activities and refinements that have been pointed out explicitly in recent publications.

Second, the *extractive development process* (right ↓) is a first extension to the baseline process models. We actually found different instantiations of this process, both usually starting with diffing of artifacts. On the one hand, an organization can decide to perform a full-fledged feature-oriented integration, meaning that it performs the same analyses (but focused on variability first [76]), scoping, and variability modeling as in the proactive adoption. This represents a top-down approach for extracting the product line. However, after the modeling, an organization usually designs an architecture by extracting and adapting an architecture from the existing variants. Afterwards, the refactoring mainly includes locating the identified features as well as adapting their assets to the architecture and adding variation points.

On the other hand, an organization may decide to simply integrate variants without defining the platform first. Instead, the platform is built by refactoring the integrated variants, which involves identifying and locating features as well as adapting the corresponding assets (e.g., adding variation points, improving re-usability)—representing a bottom-up approach. For managing the product line, the organization must model the refactored variability in parallel. As for the proactive adoption, in both instantiations the organization also has to map artifacts before the platform can be tested and eventually released. However, especially for the second instantiation, the organization may iteratively integrate variants, resulting in a loop.

As we can see, the extractive development process comprises similar activities as the proactive one. Still, there are differences in these activities, for example, in the refactoring of the platform, adapting assets, and the missing domain analysis (i.e., the variants are already established in the domain). Moreover, if an organization does not employ a feature-oriented integration, the process and its order of activities vary considerably.

Third, the *reactive adoption process* is not mentioned in the publications we analyzed. However, this is rather unproblematic, since reactive adoption is only a special case of the *variant-based evolution*. Particularly, a first variant is implemented without the platform, and can afterwards be extended by integrating new assets or variants into the first one. So, promote-pl represents all adoption strategies, and especially for the reactive adoption process we can see that domain and application engineering are mingled.

Product-Line Evolution. The evolution of a product line is driven by new customer requirements. So, while we distinguish between three different evolution processes, they usually start with the development of a new variant, and the typical application-engineering activities used for requirements analysis and scoping the variant. However, after understanding what new assets are required for developing a variant (i.e., during its design) and deciding to reuse the platform, the individual evolution processes differ.

First, *platform-based evolution* (left ↻) is typically assumed implicitly in baseline process models (cf. Fig. 2). Thus, the evolution at this point switches from application- to domain-engineering activities. The new asset must be proposed to the platform, designed to fit the platform architecture, implemented, which also includes adding variation points and modeling the new variability, tested, and integrated. To this end, an organization may use feature forks, but the core concept is a fast or continuous integration and close coordination with the platform. Afterwards, the variant can be derived by selecting its features, defining a configuration, and identifying the corresponding assets for integrating them into a repository. Identifying assets can be fully automated based on different technical solutions (e.g., configuration managers), but without such automation developers have to identify and pull the assets from different sources. Finally, the variant may require further adaptations that should not be part of the platform, or can be tested and released as is. Still, we found and experienced that organizations do not employ platform-based evolution, but, instead, rely on the following processes.

Second, during *variant-based evolution using asset propagation* (right ↻), an organization derives and clones a variant from its existing platform that is close to the new variant. In some cases, this clone may even represent the complete platform, for instance, when developing highly innovative variants that may be intended to remain separated. After adapting the variant by adding new assets, the organization may find that these assets are relevant for other variants or even the whole platform. So, assets are propagated to the platform, employing a similar process as for platform-based evolution, namely implementing an asset for reuse, testing its functionality, and finally integrating it into the platform. In particular, we experienced this evolution process for established markets where variants require new assets that have a high potential for various customers, and thus are intended for integration early on. An important prerequisite for this process is that the variant has not co-evolved for too long from the platform, as this challenges asset integration (i.e., the platform may have changed too much for simply propagating the asset)—in which case the third evolution process is more likely.

Third, *variant-based evolution using variant integration* (↻) refers to the re-integration of complete variants into the platform. We found this to be a common case if variants evolved for a longer time without synchronization with the platform, for example, in the case of highly innovative variants, co-evolution resulting in clone & own, or reactive product-line adoption. However, we also found that such variants are re-integrated based on the same process as the extractive adoption: The variant is diffed and then integrated by refactoring it to fit the platform, which may involve variability analysis, scoping, and variability modeling first; or a direct integration and parallel variability model. So, we can see that variant-based evolution, particularly with variant integration, switches the typical order of domain and application engineering, first implementing a variant to then integrate the new assets into the platform.

Domain and Application Engineering. The aforementioned process descriptions already showed that domain- and application-engineering are far more tangled and exist in varying orders compared to baseline process models. So, the typical activities associated with these two phases are represented in promote-pl, but the individual processes iterate between them. Since this is based on the publications we analyzed and aligns with our experiences, it seems that these two phases are rather cross-cutting concerns in contemporary product-line adoption and evolution processes. For this reason, they are still important and helpful to structure product-line engineering, but promote-pl is an important update to provide a more comprehensive, practice-oriented, and recent overview of product-line practices.

By constructing promote-pl, we found:

- A switch in the primary concern of interest from domain and application engineering to adoption and evolution.
- That important management activities must run in parallel to the development, but seem to be less investigated in research.
- That several adaptations to previous process models were necessary to incorporate the three adoption strategies.
- That variant-based evolution via asset- or variant-integration is the major strategy to drive the evolution of a product-line.
- That domain and application engineering are rather cross-cutting instead of primary decomposition concerns.

5 SE PRACTICES (RO_3)

In this section, we discuss promote-pl's relations to contemporary, trending software-engineering practices, which are typically applied in combination.

Continuous Software Engineering. Referred to as continuous software engineering [29], modern processes increasingly aim at bringing different phases together—reflected in recent practices including continuous integration [25], continuous deployment, continuous testing, or DevOps [26]. This trend is reflected in promote-pl, bringing together domain and application engineering in an iterative, round-trip-like process. The product-line literature recently also emphasized these practices for variability management [34, 83], and we experienced the demand for respective tool and methodological support first-hand with industrial partners [13].

When engineering variant-rich systems, continuous software engineering requires a configurable (product-line) platform. For instance, continuous deployment requires automated configuration, since manually assembling the final system (i.e., variant) cannot be done manually, or using clone & own for frequent (continuous) deployment. Likewise, continuous integration facilitates evolving the trunk using short-lived clones, and continuous testing also requires automated configuration for running test cases.

In this light, *promote-pl resolves a discrepancy between continuous software engineering and the pre-dominant extractive and reactive adoption strategies* [11] of product lines. It supports adopting a platform extractively or reactively, and evolving the platform via variants. The latter, depending on the extent of architectural deviation from the platform (see the activity *Release Variant* with its decision nodes *Integrate Asset* or *Integrate Variant* in Fig. 3), can be integrated with the same activities as adopting a platform extractively (deviation) or in a more continuous-integration-like way (no deviation). This aspect of promote-pl unifies evolution and re-engineering, and establishes round-trip engineering.

Clone Management and Incremental Adoption of Platforms. Organizations primarily use clone & own to implement variants [11, 13, 51], which is a cheap and readily available strategy, typically based on using branching facilities in version-control systems [24, 98, 99]. However, the maintenance effort for cloned variants can easily explode. To support evolution [100] before investing in a platform, clone-management frameworks strive to help synchronizing variants and keeping an overview understanding [2, 72, 81, 86, 87, 89]. A step towards clone management are governance strategies for branching and merging [24]—explicit rules for engineers when creating variant branches, aiming at reducing maintenance overhead to some extent. Staples and Hill [99], for instance, provide a branching model, which is also instantiated elsewhere [3]. However, with an increasing number of variants, it may still be necessary to adopt a platform. Instead of big-bang efforts, recently, incremental adoption strategies have been proposed [2, 28], aiming at incremental benefits for incremental investments, and therefore avoiding the risks of big-bang migrations, which disrupt development and the ability to sell products [17, 37, 55, 91]. Finally, another common practice is to use concepts of a configurable platform (e.g., variation points) together with clone & own [13]. In this light, *promote-pl explicitly supports clone management as well as an incremental adoption of platforms, or using both in a unified manner.*

Dynamic Configuration and Adaptive Systems. Modern, adaptive systems require late and dynamic binding, including microservice [101], cyber-physical [106], industry 4.0 [66], and cloud computing systems [23]. There, resource variations, asset availability, and environmental changes require systems and their software to adapt at runtime. To this end, a platform with variability as well as parameterization mechanisms needs to be adopted. A difference is that such platforms are not necessarily variant-rich systems. Instead, parameterization allows tuning or customizing systems to specific needs at runtime. Not surprisingly, several of our analyzed publications describe product-line engineering in such contexts [16, 57, 68, 108]. In this light, *the adoption strategy is rather reactive, where a single system is developed and gradually extended with variation points, as covered in promote-pl.* Still, better methods and tools are needed to manage and evolve dynamic and adaptive platforms [4, 59].

Agile Practices. Agile software engineering [70, 73] methodologies focus on customer involvement, small increments, and fast feedback. Almost all agile methodologies also build on the notion of features, including SCRUM, XP, and FDD (feature-driven development). They also foster automated testing, which, similar to continuous software engineering, requires configurable platforms.

We found two publications that report to adapt agile methods for their product-line engineering: Fogdal et al. [30] underpin that product-line and agile engineering are not conflicting, but the developers must be aware that they deliver assets to a platform that is used by others. Slightly in contrast, Hayashi et al. [36] find that agile methods are not ideal to *Derive Variants*, because the development cycles are too short to *Train Developers*. However, they adapted agile methods for evolving and throughout multiple product lines, which facilitated their engineering. These experiences indicate that agile methods are important, particularly to *Evolve* a product line (e.g., via its variants). In this light, *promote-pl does not only support agile practices, but is also crucial given the focus of agile methods on features, automation (similar to continuous software engineering), and incremental evolution via variants.*

Simulation in Testing. Three of the analyzed publications, and two of our industrial partners [13], explicitly mention to use simulation environments to *Test* their platforms and variants [13, 16, 30, 39]. While promote-pl captures these activities, more research is needed in this direction. In particular, this is different to sampling variants and test them, as safety-critical systems (e.g., cars, power plants) require an actual simulation environment to test whether software and hardware interact correctly. Moreover, as the simulators may have different properties (e.g., for transferring data) or require additional features (e.g., for *Monitoring* additional data), this can also result in simulation-specific assets (*Add Variation Points*). In this light, *promote-pl covers the relevant activities, and can guide the development of supporting techniques for simulation testing.*

6 THREATS TO VALIDITY

Construct Validity. Regarding construct validity, we may have misinterpreted the terminology used in different publications. Even more, some sets of activities have been used in varying orders in different publications, indicating variations in the use of the constructs we investigated. As a result, the orders of activities we elicited may not completely represent those intended by the authors. We mitigated this threat by building on 33 publications, carefully reading the descriptions of activities in each publication, and reasoning based on our experiences.

Internal Validity. Our work may be threatened by the methodology we employed. We may have falsely disregarded publications, missed important data during the extraction, and not derived the most suitable process model—particularly as we also relied on our experiences and interpretation. However, to limit these threats, we adapted recommendations for process theory [85], suggesting that secondary studies (e.g., systematic literature reviews) are a reliable source for such studies to reduce the potential bias of personal knowledge. Moreover, we were careful to not overly interpret the data we elicited, and checked all outcomes among all authors.

External Validity. The goal of this study was not to derive a universal process theory, which is arguably not possible. Instead, we aimed to capture how product-line engineering is currently done based on empirical data. So, as our data also shows, several process properties, such as the technologies used, the domain of the product line, and the developers involved, limit the transfer of our results to other organizations. We mitigated this external threat by considering various publications and reflecting on our industrial collaborations. For this reason, we argue that we mitigated this threat as far as possible, considering our goal of analyzing current practices.

Conclusion Validity. Other researchers may derive a different process model for product-line engineering, depending on the publications they consider, their experiences, or the construction process. To limit this threat to the conclusion validity, we explained our methodology, reasoned about our modeling decisions, and documented all publications we considered. Thus, we enable researchers to replicate and verify promote-pl.

7 RELATED WORK

Software reuse, its methods, technologies, and processes have been studied extensively. Related to our work, de Almeida et al. [22] survey existing domain-engineering and product-line engineering processes. Similarly, Frakes and Kang [31] as well as Krueger [47] provide general overviews on software reuse, including adoption strategies, methodologies, and techniques. In contrast to us, none of these works derives a process model, and they are rather old—missing insights on current practices and technological advances.

Several researchers, including ourselves, have analyzed how developers reuse software in practice. For instance, we investigated feature-modeling practices [11, 76] in order to understand how feature models are adopted and constructed in practice, but this is only one activity in the process model. Van der Lindern et al. [104] report 10 experience reports of how organizations employ product-line engineering, and what benefits they achieved. However, these cases are comparably old and analyzed in the context of the process model of Pohl et al. [82]. In a similar direction Bauer and Vetro' [6] compare the reuse practices of two organizations, but do not derive a process model for these. We reviewed other related work, which describes (parts of) product-line engineering processes based on practical experiences, to construct promote-pl (cf. Tbl. 1).

Out of the numerous literature studies on product-line engineering [15], the works of Laguna and Crespo [62], Fenske et al. [27], and Assunção et al. [5] may be the closest to promote-pl. Laguna and Crespo perform a systematic mapping study on product-line evolution, identifying 23 studies on extractive processes. Fenske et al. build on that study, including some additional papers based on their selection to derive a taxonomy (which is similar to a process theory [85]) of product-line re-engineering. Most recently, Assunção et al. report a systematic literature review, also on re-engineering. In contrast to the other two papers, the authors synthesize a high-level process model (see the corresponding partial order in Tbl. 1). All of these works focus on the specific processes of re-engineering product lines, which is part of promote-pl. So, these works are complementary, and they actually argue that well-defined, contemporary process models are needed; which we contribute with promote-pl.

8 CONCLUSION

We presented promote-pl, a modern process model for product-line engineering. Its design is based on a systematic analysis of the literature (experience reports and empirical studies) and our own industrial experiences. We adapted a method for deriving process theories to identify engineering activities and their (partial) orders as reported in the literature, and then unified the terminology to create an aggregated process model. The granularity of promote-pl allows practitioners to easily map and apply activities to various development processes—less strictly than existing process models.

Core characteristics of promote-pl, as opposed to existing process models, which were conceived almost two decades ago, are the:

- focus on **adoption and evolution strategies** as the dominant decomposition criteria of the process, which is more aligned with primary organizational concerns;
- support for **different adoption strategies**, including the dominant extractive and reactive platform adoptions;
- support to **evolve a platform via its variants** instead of primarily via the platform itself; and
- **alignment with modern practices** including continuous software engineering, agile methods, clone management, incremental platform adoption, and simulation-based testing.

We envision that future research will investigate the adoption of promote-pl in case studies, and build corresponding tool support. Also, we hope to inspire practitioners providing experience reports and requirements for tools supporting promote-pl.

Acknowledgments. Supported by DFG (SA 465/49-3), Vetenskapsrådet (257822902), DAAD, and the Wallenberg Academy.

REFERENCES

[1] Muhammad S. Ali, Muhammad A. Babar, and Klaus Schmid. 2009. A Comparative Survey of Economic Models for Software Product Lines. In *SEAA*. IEEE.

[2] Michał Antkiewicz, Wenbin Ji, Thorsten Berger, Krzysztof Czarnecki, Thomas Schmorleiz, Ralf Lämmel, Stefan Stănciulescu, Andrzej Wąsowski, and Ina Schäfer. 2014. Flexible Product Line Engineering with a Virtual Platform. In *ICSE*. ACM.

[3] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer.

[4] Wesley K. G. Assunção, Jacob Krüger, and Willian D. F. Mendonça. 2020. Variability Management meets Microservices: Six Challenges of Re-Engineering Microservice-Based Webshops. In *SPLC*. ACM.

[5] Wesley K. G. Assunção, Roberto E. Lopez-Herrejon, Lukas Linsbauer, Silvia R. Vergilio, and Alexander Egyed. 2017. Reengineering Legacy Applications into Software Product Lines: A Systematic Mapping. *Empirical Software Engineering* 22, 6 (2017).

[6] Veronika Bauer and Antonio Vetro'. 2016. Comparing Reuse Practices in Two Large Software-Producing Companies. *Journal of Systems and Software* 117 (2016).

[7] David Benavides, Sergio Segura, and Antonio Ruiz-Cortés. 2010. Automated Analysis of Feature Models 20 Years Later: A Literature Review. *Information Systems* 35, 6 (2010).

[8] Thorsten Berger, Marsha Chechik, Timo Kehrer, and Manuel Wimmer (Eds.). 2019. *Software Evolution in Time and Space: Unifying Version and Variability Management*. Schloss Dagstuhl.

[9] Thorsten Berger, Daniela Lettner, Julia Rubin, Paul Grünbacher, Adeline Silva, Martin Becker, Marsha Chechik, and Krzysztof Czarnecki. 2015. What is a Feature? A Qualitative Study of Features in Industrial Software Product Lines. In *SPLC*. ACM.

[10] Thorsten Berger, Divya Nair, Ralf Rublack, Joanne M. Atlee, Krzysztof Czarnecki, and Andrzej Wasowski. 2014. Three Cases of Feature-Based Variability Modeling in Industry. In *MODELS*. Springer.

[11] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *VaMoS*. ACM.

[12] Thorsten Berger, Stefan Stănciulescu, Ommund Ogaard, Oystein Haugen, Bo Larsen, and Andrzej Wąsowski. 2014. To Connect or Not to Connect: Experiences from Modeling Topological Variability. In *SPLC*. IEEE.

[13] Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, and Jabier Martinez. 2020. The State of Adoption and the Challenges of Systematic Variability Management in Industry. *Empirical Software Engineering* 25, 3 (2020).

[14] Danilo Beuche. 2004. Variants and Variability Management with pure::variants. In *SPLC*.

[15] Marimuthu C and K. Chandrasekaran. 2017. Systematic Studies in Software Product Lines: A Tertiary Study. In *SPLC*. ACM.

[16] Rafael Capilla and Jan Bosch. 2016. Dynamic Variability Management Supporting Operational Modes of a Power Plant Product Line. In *VaMoS*. ACM.

[17] Paul C. Clements and Charles W. Krueger. 2002. Point/Counterpoint: Being Proactive Pays Off / Eliminating the Adoption Barrier. *IEEE Software* 19, 4 (2002).

[18] Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, Ángeles S. Places, and Jennifer Pérez. 2017. Web-Based Geographic Information Systems SPLE: Domain Analysis and Experience Report. In *SPLC*. ACM.

[19] Bill Curtis, Marc I. Kellner, and Jim Over. 1992. Process Modeling. *Communications of the ACM* 35, 9 (1992).

[20] Krzysztof Czarnecki. 2004. Overview of Generative Software Development. In *UPP*. Springer.

[21] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *VaMoS*. ACM.

[22] Eduardo S. de Almeida, Alexandre Alvaro, Daniel Lucrédio, Vinicius C. Garcia, and Silvio R. de Lemos Meira. 2005. A Survey on Software Reuse Processes. In *IRI*. IEEE.

[23] Tharam Dillon, Chen Wu, and Elizabeth Chang. 2010. Cloud Computing: Issues and Challenges. In *AINA*. IEEE.

[24] Yael Dubinsky, Julia Rubin, Thorsten Berger, Slawomir Duszynski, Martin Becker, and Krzysztof Czarnecki. 2013. An Exploratory Study of Cloning in Industrial Software Product Lines. In *CSMR*. IEEE.

[25] Paul M. Duvall, Steve Matyas, and Andrew Glover. 2007. *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson.

[26] Christof Ebert, Gorka Gallardo, Josune Hernantes, and Nicolas Serrano. 2016. DevOps. *IEEE Software* 33, 3 (2016).

[27] Wolfram Fenske, Thomas Thüm, and Gunter Saake. 2013. A Taxonomy of Software Product Line Reengineering. In *VaMoS*. ACM.

[28] Stefan Fischer, Lukas Linsbauer, Roberto E. Lopez-Herrejon, and Alexander Egyed. 2014. Enhancing Clone-and-Own with Systematic Reuse for Developing Software Variants. In *ICSME*. IEEE.

[29] Brian Fitzgerald and Klaas-Jan Stol. 2014. Continuous Software Engineering and Beyond: Trends and Challenges. In *RCoSE*. ACM.

[30] Thomas Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. 2016. Ten Years of Product Line Engineering at Danfoss: Lessons Learned and Way Ahead. In *SPLC*. ACM.

[31] William B. Frakes and Kyo Kang. 2005. Software Reuse Research: Status and Future. *IEEE Transactions on Software Engineering* 31, 7 (2005).

[32] Cristina Gacek and Michalis Anastasopoules. 2001. Implementing Product Line Variabilities. In *SSR*. ACM.

[33] Sergio Garcia, Daniel Strueber, Davide Brugali, Alessandro Di Fava, Philipp Schillinger, Patrizio Pelliccione, and Thorsten Berger. 2019. Variability Modeling of Service Robots: Experiences and Challenges. In *VaMoS*. ACM.

[34] Susan P. Gregg, Denise M. Albert, and Paul C. Clements. 2017. Product Line Engineering on the Right Side of the "V". In *SPLC*. ACM.

[35] Kengo Hayashi and Mikio Aoyama. 2018. A Multiple Product Line Development Method Based on Variability Structure Analysis. In *SPLC*. ACM.

[36] Kengo Hayashi, Mikio Aoyama, and Keiji Kobata. 2017. Agile Tames Product Line Variability: An Agile Development Method for Multiple Product Lines of Automotive Software Systems. In *SPLC*. ACM.

[37] William A. Hetrick, Charles W. Krueger, and Joseph G. Moore. 2006. Incremental Return on Incremental Investment: Engenio's Transition to Software Product Line Practice. In *OOPSLA*. ACM.

[38] Jose-Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2019. Software Product Line Engineering: A Practical Experience. In *SPLC*. ACM.

[39] Takahiro Iida, Masahiro Matsubara, Kentaro Yoshimura, Hideyuki Kojima, and Kimio Nishino. 2016. PLE for Automotive Braking System with Management of Impacts from Equipment Interactions. In *SPLC*. ACM.

[40] Salma Imtiaz, Muneera Bano, Naveed Ikram, and Mahmood Niazi. 2013. A Tertiary Study: Experiences of Conducting Systematic Literature Reviews in Software Engineering. In *EASE*. ACM.

[41] Hans Peter Jepsen and Danilo Beuche. 2009. Running a Software Product Line: Standing still is Going Backwards. In *SPLC*. ACM.

[42] Hans Peter Jepsen, Jan Gaardsted Dall, and Danilo Beuche. 2007. Minimally Invasive Migration to Software Product Lines. In *SPLC*. IEEE.

[43] Hans Peter Jepsen and Flemming Nielsen. 2000. A Two-Part Architectural Model as Basis for Frequency Converter Product Families. In *IW-SAPF*. Springer.

[44] Kyo C. Kang, Jaejoon Lee, and Patrick Donohoe. 2002. Feature-Oriented Product Line Engineering. *IEEE Software* 19, 4 (2002).

[45] Barbara A. Kitchenham, David Budgen, and Pearl Brereton. 2015. *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press.

[46] Heiko Koziolek, Thomas Goldschmidt, Thijmen de Gooijer, Dominik Domis, Stephan Sehestedt, Thomas Gamer, and Markus Aleksy. 2016. Assessing Software Product Line Potential: An Exploratory Industrial Case Study. *Empirical Software Engineering* 21, 2 (2016).

[47] Charles W. Krueger. 1992. Software Reuse. *Comput. Surveys* 24, 2 (1992).

[48] Charles W. Krueger. 2001. Easing the Transition to Software Mass Customization. In *PFE*. Springer.

[49] Charles W. Krueger and Paul C. Clements. 2013. Systems and Software Product Line Engineering with BigLever Software Gears. In *SPLC*. ACM.

[50] Jacob Krüger. 2016. *A Cost Estimation Model for the Extractive Software-Product-Line Approach*. Master's thesis. University of Magdeburg.

[51] Jacob Krüger. 2019. Are You Talking about Software Product Lines? An Analysis of Developer Communities. In *VaMoS*. ACM.

[52] Jacob Krüger, Sofia Ananieva, Lea Gerling, and Eric Walkingshaw. 2020. Third International Workshop on Variability and Evolution of Software-Intensive Systems (VariVolution 2020). In *SPLC*. ACM.

[53] Jacob Krüger and Thorsten Berger. 2020. Activities and Costs of Re-Engineering Cloned Variants into an Integrated Platform. In *VaMoS*. ACM.

[54] Jacob Krüger, Thorsten Berger, and Thomas Leich. 2019. *Software Engineering for Variability Intensive Systems*. CRC Press, Chapter Features and How to Find Them: A Survey of Manual Feature Location.

[55] Jacob Krüger, Wolfram Fenske, Jens Meinicke, Thomas Leich, and Gunter Saake. 2016. Extracting Software Product Lines: A Cost Estimation Perspective. In *SPLC*. ACM.

[56] Jacob Krüger, Christian Lausberger, Ivonne von Nostitz-Wallwitz, Gunter Saake, and Thomas Leich. 2020. Search. Review. Repeat? An Empirical Study of Threats to Replicating SLR Searches. *Empirical Software Engineering* 25, 1 (2020).

[57] Jacob Krüger, Mukelabai Mukelabai, Wanzi Gu, Hui Shen, Regina Hebig, and Thorsten Berger. 2019. Where is my Feature and What is it About? A Case Study on Recovering Feature Facets. *Journal of Systems and Software* 152 (2019).

[58] Jacob Krüger, Louis Nell, Wolfram Fenske, Gunter Saake, and Thomas Leich. 2017. Finding Lost Features in Cloned Systems. In *SPLC*. ACM.

[59] Jacob Krüger, Sebastian Nielebock, Sebastian Krieter, Christian Diedrich, Thomas Leich, Gunter Saake, Sebastian Zug, and Frank Ortmeier. 2017. Beyond Software Product Lines: Variability Modeling in Cyber-Physical Systems. In *SPLC*. ACM.

[60] Jacob Krüger and Thorsten Berger. 2020. An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In *ESEC/FSE*. ACM.

[61] Elias Kuiter, Jacob Krüger, Sebastian Krieter, Thomas Leich, and Gunter Saake. 2018. Getting Rid of Clone-and-Own: Moving to a Software Product Line for Temperature Monitoring. In *SPLC*. ACM.

[62] Miguel A. Laguna and Yania Crespo. 2013. A Systematic Mapping Study on Software Product Line Evolution: From Legacy System Reengineering to Product Line Refactoring. *Science of Computer Programming* 78, 8 (2013).

[63] Max Lillack, Ştefan Stănciulescu, Wilhelm Hedman, Thorsten Berger, and Andrzej Wąsowski. 2019. Intention-Based Integration of Software Variants. In *ICSE*. IEEE.

[64] Robert Lindohf, Jacob Krüger, Erik Herzog, and Thorsten Berger. 2020. Software Product-Line Evaluation in the Large. *Empirical Software Engineering* (2020).

[65] Lukas Linsbauer, Somayeh Malakuti, Andrey Sadovykh, and Felix Schwägerl. 2018. 1st Intl. Workshop on Variability and Evolution of Software-Intensive Systems (VariVolution). In *SPLC*. ACM.

[66] Yang Lu. 2017. Industry 4.0: A Survey on Technologies, Applications and Open Research Issues. *Journal of Industrial Information Integration* 6 (2017).

[67] Luciano Marchezan, Elder Macedo Rodrigues, Maicon Bernardino, and Fábio Paulo Basso. 2019. PAxSPL: A Feature Retrieval Process for Software Product Line Reengineering. *Software: Practice and Experience* 49, 8 (2019).

[68] Jabier Martinez, Xhevahire Tërnava, and Tewfik Ziadi. 2018. Software Product Line Extraction from Variability-Rich Systems: The Robocode Case Study. In *SPLC*. ACM.

[69] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer.

[70] Bertrand Meyer. 2014. *Agile!* Springer.

[71] Mathias Meyer. 2014. Continuous Integration and its Tools. *IEEE Software* 31, 3 (2014).

[72] Leticia Montalvillo and Oscar Díaz. 2015. Tuning GitHub for SPL Development: Branching Models & Repository Operations for Product Engineers. In *SPLC*. ACM.

[73] Alan Moran. 2015. *Managing Agile*. Springer.

[74] Mukelabai Mukelabai, Damir Nesic, Salome Maro, Thorsten Berger, and Jan-Philipp Steghöfer. 2018. Tackling Combinatorial Explosion: A Study of Industrial Needs and Practices for Analyzing Highly Configurable Systems. In *ASE*. ACM.

[75] Motoi Nagamine, Tsuyoshi Nakajima, and Noriyoshi Kuno. 2016. A Case Study of Applying Software Product Line Engineering to the Air Conditioner Domain. In *SPLC*. ACM.

[76] Damir Nešić, Jacob Krüger, Stefan Stănciulescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *ESEC/FSE*. ACM.

[77] Michael Nieke, Lukas Linsbauer, Jacob Krüger, and Thomas Leich. 2019. Second International Workshop on Variability and Evolution of Software-Intensive Systems (VariVolution 2019). In *SPLC*. ACM.

[78] Andy J. Nolan and Silvia Abrahão. 2010. Dealing with Cost Estimation in Software Product Lines: Experiences and Future Directions. In *SPLC*. Springer.

[79] Linda M. Northrop. 2002. SEI's Software Product Line Tenets. *IEEE Software* 19, 4 (2002).

[80] Object Management Group. 2007. *Unified Modeling Language: Superstructure Version 2.1.1*.

[81] Tristan Pfofe, Thomas Thüm, Sandro Schulze, Wolfram Fenske, and Ina Schaefer. 2016. Synchronizing Software Variants with VariantSync. In *SPLC*. ACM.

[82] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering*. Springer.

[83] Richard Pohl, Mischa Höchsmann, Philipp Wohlgemuth, and Christian Tischer. 2018. Variant Management Solution for Large Scale Software Product Lines. In *ICSE-SEIP*. ACM.

[84] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. 2018. A Study and Comparison of Industrial vs. Academic Software Product Line Research Published at SPLC. In *SPLC*. ACM.

[85] Paul Ralph. 2019. Toward Methodological Guidelines for Process Theories and Taxonomies in Software Engineering. *Transactions on Software Engineering* 45, 7 (2019).

[86] Julia Rubin and Marsha Chechik. 2013. A Framework for Managing Cloned Product Variants. In *ICSE*. IEEE.

[87] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2013. Managing Cloned Variants: A Framework and Experience. In *SPLC*. ACM.

[88] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2015. Cloned Product Variants: From Ad-Hoc to Managed Software Product Lines. *International Journal on Software Tools for Technology Transfer* 17, 5 (2015).

[89] Julia Rubin, Andrei Kirshin, Goetz Botterweck, and Marsha Chechik. 2012. Managing Forked Product Variants. In *SPLC*. ACM.

[90] Mohammed Sayagh, Noureddine Kerzazi, Bram Adams, and Fabio Petrillo. 2018. Software Configuration Engineering in Practice: Interviews, Survey, and Systematic Literature Review. *IEEE Transactions on Software Engineering* (2018).

[91] Klaus Schmid and Martin Verlage. 2002. The Economic Impact of Product Line Adoption and Evolution. *IEEE Software* 19, 4 (2002).

[92] Pierre-Yves Schobbens, Patrick Heymans, and Jean-Christophe Trigaux. 2006. Feature Diagrams: A Survey and a Formal Semantics. In *RE*. IEEE.

[93] Christoph Seidl, Thorsten Berger, Christoph Elsner, and Klaus-Benedikt Schultis. 2017. Challenges and Solutions for Opening Small and Medium-Scale Industrial Sofware Platforms. In *SPLC*. ACM.

[94] Yusra Shakeel, Jacob Krüger, Ivonne von Nostitz-Wallwitz, Christian Lausberger, Gabriel Campero Durand, Gunter Saake, and Thomas Leich. 2018. (Automated) Literature Analysis - Threats and Experiences. In *SE4Science*. ACM.

[95] Dag I. K. Sjøberg, Tore Dybå, Bente C. D. Anda, and Jo E. Hannay. 2008. Building Theories in Software Engineering. In *Guide to Advanced Empirical Software Engineering*. Springer.

[96] Hannah Snyder. 2019. Literature Review as a Research Methodology: An Overview and Guidelines. *Journal of Business Research* 104 (2019).

[97] Daniel Ståhl, Kristofer Hallén, and Jan Bosch. 2016. Continuous Integration and Delivery Traceability in Industry: Needs and Practices. In *SEAA*. IEEE.

[98] Stefan Stănciulescu, Sandro Schulze, and Andrzej Wasowski. 2015. Forked and Integrated Variants in an Open-Source Firmware Project. In *ICSME*. IEEE.

[99] Mark Staples and Derrick Hill. 2004. Experiences Adopting Software Product Line Development Without a Product Line Architecture. In *APSEC*. IEEE.

[100] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems. In *SPLC*. ACM.

[101] Johannes Thönes. 2015. Microservices. *IEEE Software* 32, 1 (2015).

[102] Thomas Thüm, Sven Apel, Christian Kästner, Ina Schaefer, and Gunter Saake. 2014. A Classification and Survey of Analysis Strategies for Software Product Lines. *Comput. Surveys* 47, 1 (2014).

[103] Muhammad Usman, Muhammad Zohaib Iqbal, and Muhammad Uzair Khan. 2017. A Product-Line Model-Driven Engineering Approach for Generating Feature-Based Mobile Applications. *Journal of Systems and Software* 123 (2017).

[104] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action*. Springer.

[105] Jens H. Weber, Anita Katahoire, and Morgan Price. 2015. Uncovering Variability Models for Software Ecosystems from Multi-Repository Structures. In *VaMoS*. ACM.

[106] Wayne Wolf. 2009. Cyber-Physical Systems. *Computer* 3 (2009).

[107] Bobbi Young, Judd Cheatwood, Todd Peterson, Rick Flores, and Paul Clements. 2017. Product Line Engineering Meets Model Based Engineering in the Defense and Automotive Industries. In *SPLC*. ACM.

[108] Tao Yue, Shaukat Ali, and Bran Selic. 2015. Cyber-Physical System Product Line Engineering: Comprehensive Domain Analysis and Experience Report. In *SPLC*. ACM.