Software Product Line Engineering and Variability Management: Achievements and Challenges

Andreas Metzger
paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen
45127 Essen, Germany
andreas.metzger@paluno.uni-due.de

Klaus Pohl
paluno (The Ruhr Institute for Software Technology)
University of Duisburg-Essen
45127 Essen, Germany
klaus.pohl@paluno.uni-due.de

ABSTRACT
Software product line engineering has proven to empower organizations to develop a diversity of similar software-intensive systems (applications) at lower cost, in shorter time, and with higher quality when compared with the development of single systems. Over the last decade the software product line engineering research community has grown significantly. It has produced impressive research results both in terms of quality as well as quantity. We identified over 600 relevant research and experience papers published within the last seven years in established conferences and journals. We briefly summarize the major research achievements of these past seven years. We structure this research summary along a standardized software product line framework. Further, we outline current and future research challenges anticipated from major trends in software engineering and technology.

Categories and Subject Descriptors
D.2.10 **[Software Engineering]**: Design – *methodologies*; D.2.11 **[Software]**: Software Architectures – *domain-specific architectures;* D.2.13 **[Software Engineering]**: Reusable Software – *domain engineering, reuse models*

General Terms
Management, Documentation, Design, Economics, Verification

Keywords
Software product lines, requirements engineering, design, quality assurance, variability management, variability modeling

1. INTRODUCTION
Many industry sectors are faced with increasing demand to develop individualized software-intensive systems. Software product line engineering (SPLE) has proven to empower organizations to develop a diversity of similar systems at lower cost, in shorter time, and with higher quality when compared with the development of single systems [1]. A software product line (also sometimes called software product family) is "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets [artifacts] in a prescribed way" [2].

SPLE exploits the commonalities of the systems that belong to a product line and systematically handles the variation (i.e., the differences) among those systems. *Commonality* is a property shared by all applications[1] of the product line [3]; e.g., all mobile phones allow users to make calls. Product line *variability* defines how the different applications of the product line can vary [4]. Product line applications may differ in terms of features, functional and quality requirements they fulfil; e.g., some tablet computers may include mobile broadband connectivity, others not.

In industry, the SPLE paradigm has a strong track record of success. Examples of success stories can be found in textbooks (such as [1], [2], [5]) or in the product line hall of fame[2]. The product line hall of fame has been established as part of the SPLC conference series and lists 20 success stories from companies including Boeing, Bosch, HP, Nokia, Philips, Siemens and Toshiba. Reported benefits of SPLE include "improved productivity by as much as factor 10, increased quality by as much as factor 10, decreased cost by as much as 60%, decreased labor needs by as much as 87%, decreased time to market (to field, to launch) by as much as 98%, and ability to move into new markets in months, not years"[3].

SPLE has attracted significant interest from the research community. We have surveyed the literature of the past seven years and retrieved over 600 relevant papers from established conferences and journals alone[4]. A complete summary of all achievements in the field is obviously impossible in this paper. We thus summarize highlights of the research achievements in SPLE. Specifically, we focus on concepts, techniques and methods. In addition, we briefly describe open and upcoming research challenges we anticipate from major trends in software engineering and technology.

The successful introduction of SPLE in industry requires strong (high level) management commitment. Moreover, the successful introduction of SPLE heavily depends on the implementation of adequate organizational structures and processes (see, for example, Chapter 1 in [6]). Moreover, without convincing business cases which demonstrate the return-on-investment SPLE will typically not be introduced in an organization. Due to space limitations, those aspects are not within the scope of this paper. Also, achievements in tool support, both for what concerns commercial tools and prototypes, are also not within the scope.

FOSE'14, May 31 – June 7, 2014, Hyderabad, India
ACM 978-1-4503-2865-4/14/05
http://dx.doi.org/10.1145/2593882.2593888

[1] As in most of the SPLE literature, we use the terms application and (software-intensive) system synonymously.

[2] http://splc.net/fame.html

[3] http://www.sei.cmu.edu/productlines/

[4] Our paper classification is available online at http://www.sse.uni-due.de/en/fose14/

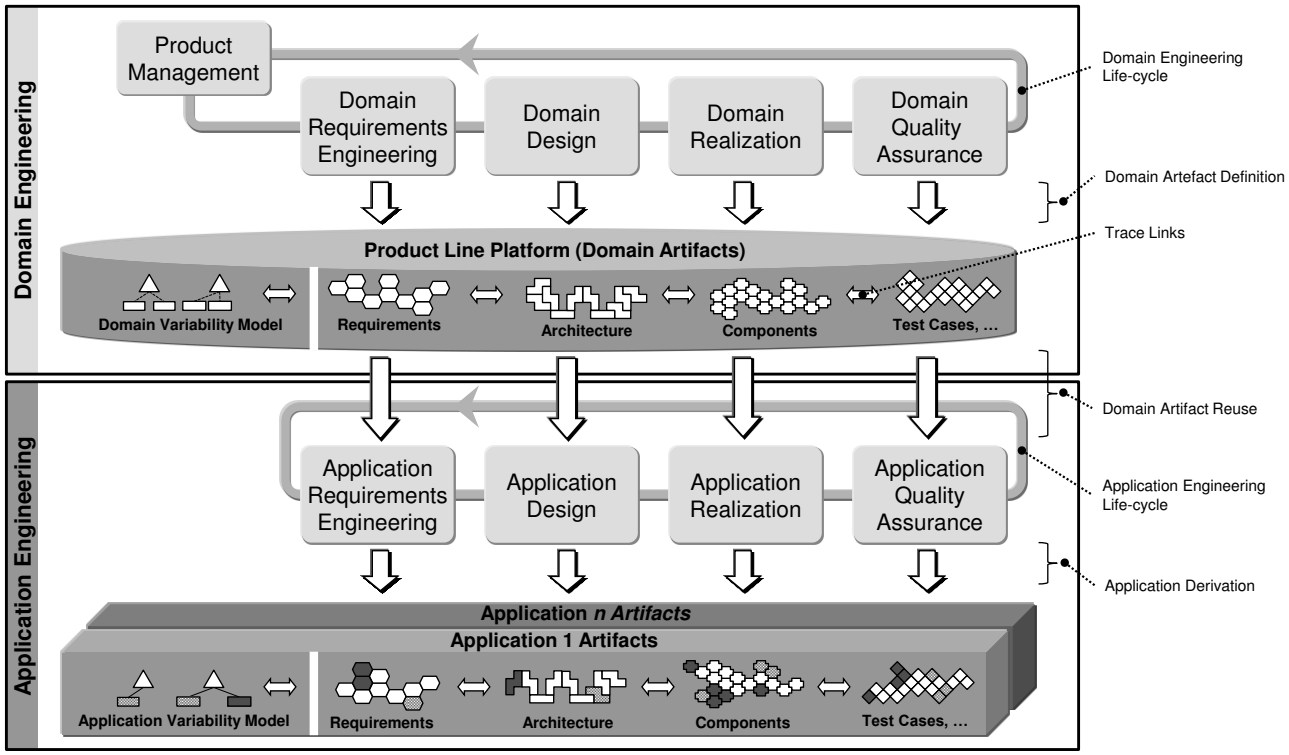


Figure 1: SPLE Framework (adapted from [1])

To structure the technical SPLE research achievements and challenges, we use a standardized framework for software product line engineering briefly described in Section 2. Following this framework, we discuss the achievements and open research challenges in the area of variability modelling and management in Section 3. Section 4 summarizes the achievements and open research challenges in the area of domain engineering. Section 5 describes the achievements and open research challenges in the area of application engineering. Major trends in software engineering and technology will lead to new research challenges. We elaborate on those challenges in Section 6.

2. FOUNDATIONS

Figure 1 depicts a well-established SPLE framework, which has recently been adopted as part of the ISO/IEC standard #26550 ("Software and systems engineering: Reference model for product line engineering and management"). This framework has been defined based on the outcomes of the European SPLE research projects ESAPS, CAFÉ, and FAMILIES. It is described in greater detail in [1]. We employ this framework in the remainder of the paper to cluster and summarize the research achievements in the field. The key elements of the framework are described below.

2.1 Two Product Line Processes

To facilitate the efficient development of a diversity of applications which share a set of commonalities, SPLE differentiates between two complementary development processes:

- The *domain engineering* process (shown in the upper half of Figure 1) is responsible for defining the commonality and the variability of the product line, as well as for developing the *domain artifacts*. Domain artifacts "realize" commonality and variability. They include, among others, requirements artifacts (e.g., use case diagrams, requirements models), architectural artifacts (e.g., component models, class diagrams) and test artifacts (e.g., test cases, test data).
- The *application engineering* process (shown in the lower half of Figure 1) is responsible for deriving concrete applications from the domain artifacts. To this end, application engineering exploits the variability of the domain artifacts by binding (resolving) the variability according to the needs and requirements for a particular application. Domain artifacts are thereby reused in application engineering to derive a set of product line applications.

By dividing the overall development process into domain engineering and application engineering, two key concerns are separated: (1) to build a robust product line platform, (2) to efficiently create individual, customer-specific or market-specific applications based on the product line platform.

The product line platform encompasses all domain artifacts of the product line.[5] Important parts of the product line platform are the domain requirements and the product line architecture, which is often called the *reference architecture* of the product line [1], [5]. The product line architecture provides a common, high-level structure for all product line applications. The domain requirements define the common and variable features, functions and qualities of the product line.

The activities executed during domain and application engineering typically do not follow a sequential ("waterfall-like") order even though the visualization of the framework depicted in Figure 1

[5] The term "platform" has slightly different meanings in other areas (e.g., see [1]).

might imply so. In general, any type of life-cycle or process model (e.g., V-model, spiral model, agile models) can be used in a software product line setting. The execution order of activities and the activities themselves therefore depend on the development process used in an organization. Moreover, like in single systems development, quality assurance activities should commence from the start of development and should accompany all activities in domain and application engineering.

Similarly, there is no strict sequence in executing the domain and the application engineering processes [7], [1]:

- In a *proactive* or "*Big Bang*" approach, domain engineering is performed completely before application engineering starts;
- In a *reactive* or *incremental* approach, the most relevant domain artifacts (typically the common ones) are developed first and variable ones are developed based on concrete customer demands during application engineering;
- In a *reengineering-driven* approach, existing individual applications and systems are "migrated" into a product line.

In most industrial cases, customer-specific applications cannot be derived entirely from the domain artifacts alone. Therefore, so called customer-specific extensions have to be implemented during application engineering [1], [8], [9].

2.2 Product Line Variability

Product line variability is the key, cross-cutting concern in SPLE [6], [10], [1], [2]. Product line variability describes the variation among the applications of a software product line in terms of properties, such as features that are offered or functional and quality requirements that are fulfilled. Whether a given property is to be common or variable across a software product line is determined by explicit management decisions, typically made by product management [4], [1]. Product line variability is documented in so-called variability models. The SPLE framework in Figure 1 differentiates between two types of variability models: Domain variability models and application variability models (cf., [1], [8]).

During domain engineering, the variability of the product line is defined in the domain variability model. In application engineering, the variability defined in the domain variability model is bound in order to fulfill the application-specific requirements. The variability bindings for a specific application are documented in a respective application variability model.

Product line variability is pre-planned in order to address the variation needed in different applications to fulfil different market and stakeholder needs. Still, application engineers may face the problem that individual customer- or market-specific needs cannot be satisfied completely by reusing common and variable domain artifacts. In this case, customer- or market-specific extensions or adjustments of the common and variable artifacts are required. The required adjustments can be made by either adjusting the domain artifacts (e.g., introducing additional product line variability) or by adaptations of the application artifacts [8].

Application-specific adjustments of artifacts should be documented in the application variability model. An application variability model thus documents both, the binding of the variability for the specific application, as well as the application-specific adaptations. An application variability model thereby establishes traceability between application and domain artifacts.

2.3 Product Line Variability vs. Software Variability

Quite often SPLE research contributions do not clearly differentiate between product line variability and software variability.

Software variability refers to the ability of software systems or artifacts to be efficiently extended, changed, customized or configured [11], [12]. Most modelling and programming languages provide mechanism for software variability. Examples include abstract super-classes allowing different specializations, interfaces facilitating different implementations, or conditional compilation (e.g., using #ifdefs) facilitating the inclusion of different code fragments. A recent, extensive survey on software variability is provided in [12].

Product line variability defines how the applications of a product line can differ. Together with the commonalities, product line variability defines the scope of a product line (see Section 4.1). Like commonalities, product line variability is pre-planned. Defining whether a given feature, functional or quality requirement is product line variability or not requires explicit decisions from product management or other stakeholders.

Software variability can represent both: product line variability as well as commonality. As an example for software variability, take the abstract super class *Communication* with two concrete sub-classes *WiFi* and *MobileBroadband* documented in a UML class diagram. Clearly, the super-class together with the sub-classes documents software variability. In principle, any of the two or even both sub-classes could be used in place of the super-class.

This software variability would represent a commonality of the product line, if – for a given product line – the stakeholders had decided that all applications must include both sub-classes *WiFi* and *MobileBroadband*. In other words, the product line applications cannot differ in terms of the communication classes they use. In this case, software variability would clearly represent a commonality of the product line (and not product line variability).

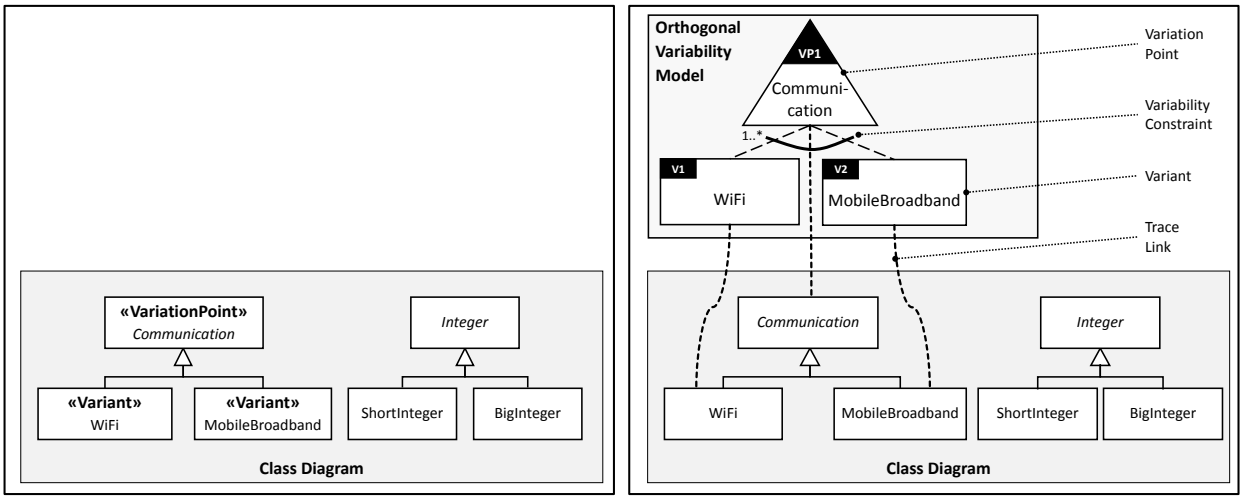
However, the same software variability could also represent product line variability. For example, if the stakeholders had decided that for each application of the product line the engineer has to choose at least one of the two sub-classes, the applications could differ in terms of the sub-classes they include for communication. In this case, the same software variability would represent a realization of product line variability.

Consequently, software variability is not sufficient to determine product line variability. In other words, product line variability cannot be identified by analyzing software variability documented in artifact models or by analyzing the software artifacts themselves.[6] The definition of product line variability requires explicit decisions. Moreover, product line variability has to be explicitly defined in addition to software variability.

3. VARIABILITY MODELING AND ANALYSIS

As sketched in Section 2.3, product line variability differs significantly from software variability. In addition to software variabil-

[6] Nevertheless, in a re-engineering setting, software variability may provide indicators for potential product line variability (e.g., see [13], [14]).



a) Integrated Variability Modelling

b) Orthogonal Variability Modelling

Figure 2: Illustration of Two Major Approaches for Explicit Variability Modeling

ity, product line variability needs to be explicitly defined to empower and support the communication, discussion, management and analysis of product line variability.

3.1 Modeling Product Line Variability

There are two principle ways in SPLE research and practice to explicitly document product line variability:

- *Integrated Documentation*: To support the integrated documentation of product line variability, dedicated or specialized modelling and documentation concepts are introduced into existing modelling languages or document templates. An example for the integrated documentation of product line variability is depicted in Figure 2a. The figure shows a UML class diagram extended by two stereotypes «VariationPoint» and «Variant». The stereotypes are used to explicitly document the product line variability. This example models a product line, in which communication is defined as product line variability (documented by *Communication* being a variation point and *WiFi* and *MobileBroadband* being variants).
- *Orthogonal Documentation*: To support the orthogonal documentation of product line variability, product line variability is documented in a dedicated model. In other words, the documentation of product line variability is separated from the documentation of the software development artifacts. Thereby the variability of the product line is treated as a first class product line artifact. By relating the product line variability defined in the orthogonal variability model with the software artifacts defined in the artifact models, the realization of product line variability within the software artifacts is documented. Figure 2b sketches an example of an orthogonal documentation of product line variability and its relation to software development artifacts. As depicted in the figure, the documentation of product line variability is clearly separated from the documentation of other software development aspects. Note, that the orthogonal variability model only defines product line variability. It does not define product line commonalities.

3.1.1 Integrated Variability Modelling

Numerous extensions of modeling languages have been proposed to facilitate the integrated documentation of product line variability, including annotations of uses cases and test models [13], stereotypes for UML diagrams (e.g., activity diagrams, state charts, or component diagrams [14], [15]), and domain-specific languages [16].

For the integrated documentation of product line variability feature models are most commonly used (e.g., see [6]). A feature model is a tree or a directed acyclic graph of features[7]. A feature model is organized hierarchically. A feature can be decomposed into sub-features. A mandatory feature has to be selected if its parent feature is mandatory or if its parent feature is optional and has been selected. Mandatory features define commonalities. Mandatory features have to be selected for all applications of the product line. Optional, alternative, and 'or' features define variability in feature models. As a result, a feature model is a compact representation of all mandatory and optional features of a software product line. Each valid combination of features represents a potential product line application.[8]

Since the introduction of Feature-Oriented Domain Analysis (FODA) by Kang et al. in the 1990ies, over 40 different feature model dialects have been proposed [17]. Based on the expressiveness of those extensions, they can be grouped into three categories: basic feature models (offering mandatory, alternative and 'or' features, as well as 'requires' and 'excludes' cross-tree constraints), cardinality-based feature models (offering, in addition, UML-like multiplicities for feature selection [*m..n*]), and extended feature models (adding arbitrary feature attributes; e.g., to express variation in quality requirements). The increased expressiveness of feature modeling languages, while appearing attractive to practitioners, has negative impact on the analyzability of the models.

[7] According to IEEE Standard 829-1998, a feature is considered „a distinguishing characteristic of a software item (e.g., performance, portability, or functionality)". In SPLE features are often considered a user-observable characteristic of a software item.

[8] In some cases, feature models are used to only document the variability of a product line. Mandatory (common) features are not documented. According to our definition, the documentation of the variability in such feature model is thus orthogonal.

Assuring the quality of a variability model thus may become difficult. As an example, the analysis of feature models (see Section 3.2) becomes more challenging when considering cardinalities [17] or it may even reach analyzability limits when using unbounded cardinality constraints or non-Boolean domains for feature attributes [18].

3.1.2 Orthogonal Variability Modelling

As introduced above, orthogonal variability models define the variability of the product line in a dedicated model. The variability defined in the orthogonal variability model is linked to elements in the software artifact models that realize product line variability. Those software artifact models are called *base models* in this context [1].

There are only a few modeling constructs used in an orthogonal variability model. Moreover, the constructs are simple. A *variation point* documents a variable item and thus defines "what can vary" (without saying how it can vary). A *variant* documents a concrete variation and is related to a variation point. A variant thus defines "how something can vary". In addition, *variability constraints* can be defined that specify restrictions about the variability; e.g., to define permissible combinations of variants in an application or to define that the selection of one variant requires or excludes the selection of another variant. The *Common Variability Language*[9], a concrete language for orthogonal variability modelling, is currently undergoing "standardization" within the OMG.

Product line variability is also (indirectly) defined in so-called decision models [19]. The aim of a decision model is to define how the variability of the software product line should be resolved during application engineering. We thus discuss decision models in Section 5.1.

3.1.3 Comparing Variability Modelling Approaches

Integrated variability modeling increases the complexity of the software artifact models and documentations due the additional documentation of product line variability within those artifacts. Moreover, product line variability is redundantly defined in different development artifacts such as requirements models, component diagrams, code, or test cases. As a result, understanding and tracing product line variability between different artifact models becomes difficult. First, different modelling constructs are used to represent the variability in the different models. As a consequence, product line variability is represented differently in the various models. Second, dependencies between the variability defined in the different artifact models are typically not documented explicitly. Third, it is difficult, if not impossible, to keep the variability defined in the different models consistent.

Orthogonal variability modelling avoids those three significant drawbacks of integrated variability modeling. In an orthogonal variability model *only* the variability of a product line is defined. Commonalities of the product line are only documented in the base models – a key difference from "traditional" feature models, which define both, commonalities and variability. The explicit differentiation between variation point and variant marks a second key difference from feature models, which do not provide explicit modeling concepts for variation points. As a third key difference, the variability definition in an orthogonal variability model is free from realization concerns. Therefore, orthogonal variability models provide a clear separation between product line variability (documented in an orthogonal variability model) and software variability (specified in the base models). When using feature models, the separation between product line variability and software variability often gets blurred [4]. Defining the variability in a dedicated, orthogonal variability model avoids this problem.

Variability defined in orthogonal variability models, as well as in feature models must be interrelated with the software development artifacts defined in the base models (e.g., see [20], [1]). Establishing and maintaining trace links between variability models and the base models is not trivial. A solution for the interrelation is to parameterize the base models to indicate which base model elements link to which feature [21]. However, this solution violates the key principle of keeping product line variability separate from base models. More recent solutions argue for dedicated mapping specifications, like in the Common Variability Language (see Section 3.1.2), which introduces mappings from variation points and variants to MOF-compliant base models.

Orthogonal variability modeling leads to less complex models compared to integrated variability modeling. Yet, given the growing size and complexity of product lines faced in industry, existing orthogonal variability modeling languages will reach their limits in handling the size and complexity of those product lines. In a recent survey, 25% of industry participants reported product lines that include more than 10,000 variants or features [22]. Early, but limited attempts for handling large-scale variability models include the use of textual languages [23] and the definition of abstraction layers for product line variability [24].

Open research challenges in **variability modelling** include:

- **Understanding tradeoffs between expressiveness and analyzability:** How to facilitate the selection of the variability modeling language that fits a given purpose? How to understand and capture the tradeoffs between expressive languages and the limitations expressiveness imposes on the analyzability and thus on the quality of the variability models [18]? The analysis of such tradeoffs should be based on realistic industrial examples and validated in sound empirical studies.
- **Interrelating variability with base models:** The solutions for interrelating variability defined in an orthogonal variability model with the software artifacts defined in the base models are still in their infancy. We need smart approaches for easy to establish and easy to maintain interrelations between variability models and software artifacts. Among others, such approaches should support consistency across the different artifacts.
- **Handling large-scale variability models:** Handling the complexity of large-scale variability models for complex systems such as cars, airplanes or power distribution networks is a key challenge in industry. Academic approaches need more industrial strength and have to become more practical and scalable. There is a wide gulf between what is published and what could reasonably be used in an industrial setting. So, how to upscale variability modeling techniques for use in industrial settings? How to define suitable variability abstraction layers or views? How to interlink variability defined in the different abstraction layers or views? How to ensure consistency between the variability defined at the different layers of abstraction, e.g., the overall car, the engine and the injection control? How to map variability defined in different variability models used in different organizations across the supply chain?

[9] http://www.omgwiki.org/variability/doku.php

3.2 Analyzing Variability

Variability analysis aims to check and ensure whether certain properties for a given variability model hold. Examples for properties to be checked are satisfiability (i.e., whether at least one application can be derived from the variability model), membership (i.e., whether a given configuration is consistent with the variability model and thus represents a valid application of the product line), commonality (i.e., the set of "features" that appear in all applications), and "dead" features (i.e., features that cannot be selected for any application).

Manual analysis of variability models is error-prone and infeasible when facing large-scale variability models. A broad spectrum of automated variability analysis techniques has thus been proposed. They can be categorized in three main classes [25], [17]: propositional-logics-based (using SAT or BDD solvers), constraint-programming-based (using CSP solvers) and description-logics-based (using DL reasoners). In general, variability model analyses exhibit an exponential worst-case execution time. Yet, research results indicate that in most cases variability model analysis can be mastered quite successfully using powerful solvers [26].

Over the last years, the SPLE community has collected a large set of variability models. These models are publically available and may be used for empirical studies and as benchmark for variability analysis. For example, the SPLOT[10] repository contains over 400 feature models from both industry and academia.

Open research challenges in **variability analysis** include:

- **Metrics for performance prediction:** How to predict the actual performance of analysis techniques for a given variability model? What are appropriate metrics to measure the structural complexity or size of variability models? Can empirical relations – based on such metrics – be established between the structure of the variability model and the resources required by a solver? How to leverage performance prediction to select the best solver for analyzing certain characteristics for a given variability model? How to develop SPLE-specific heuristics for further improving analysis performance?
- **Large-scale, realistic variability models:** Very few large-scale variability models from industrial practice are publically available, even though many such models exist [22]. Most of the large-scale variability models available (e.g., in the SPLOT repository) have been generated randomly or by mimicking properties of realistic but rather small models found in the literature [17]. Using further real-world variability models as benchmarks (even in an anonymized form) is essential for establishing empirical evidence (in addition to theoretical) for variability modeling and analysis techniques. Making large-scale, real-world variability models available is thus still an open issue.

4. DOMAIN ENGINEERING

4.1 Product Management

The main task of product management in SPLE is product line scoping [27]. One facet of product line scoping is the definition of the product portfolio, i.e., the set of applications offered for a certain market segment by a particular business unit or company. Further facets commonly include the definition of which set of features, as well as which set of domain artifacts can be economically reused [28], [29]. If the scope of a software product line is defined too broadly, domain artifacts may become too generic and the effort of realizing them may become too high. As a consequence, the product line may not be economically viable. On the other hand, if the scope is defined too narrow required features, functional and quality requirements of many customers may not be covered and thus only very few application might be derivable from the product line. Also in this case, the product line may not be economically viable. Therefore, scoping techniques need to include techniques for estimating costs and benefits, thereby enabling the optimization of the product portfolio. Scoping activities need to involve business as well as technical experts.

In many organizations multiple product lines with shared and individual properties exist. For example, within Phillips product-lines are structured according to divisions, business units, and business lines [30]. Automotive manufacturers structure their product lines according to lines, body types and countries [31]. Some initial approaches have been proposed for coordinating multi-level product lines with regard to their commonalities and variability [24], [32].

Software ecosystems constitute a recent development to generalize the notion of multi-software product lines. Software ecosystems open up software product lines to external developers to extend and use the product line platform or even extend the applications released by the product line owner [33], [6].

Open research challenges in **product management** include:

- **Scope optimization:** How to optimize the scope of a product line with regard to features and domain artifacts [29]? Besides considering economic and market aspects, scoping has to consider technical aspects like the life-cycle management of features and their realization in domain artifacts (including architectural complexity) and has to take the economic viability of the whole product line and its artifacts into account.
- **Artifact-interrelations in multi-level product lines:** How to establish trace links across multi-level product line hierarchies and their artifacts? Existing solutions have addressed individual artifact types, such as requirements or components. How to manage trace links across all product line artifacts in a hierarchical product line setting? How to bi-directionally synchronize those trace links and the artifacts, e.g., in case of changes and evolution of artifacts in a sub-line? How to propagate variability constraints, which arise in a sub-product line, to a higher level product line? How to align the product lines of suppliers and vendors? How to leverage management techniques for multi-level product lines to handle software ecosystems?

4.2 Domain Requirements Engineering

Domain requirements engineering encompasses all activities for eliciting, negotiating, documenting, validating, and managing the common and variable requirements for the product portfolio envisioned by product management. To identify all relevant common and variable requirements, product line requirements engineers have to involve a larger number of stakeholders than for single systems and have to consider additional requirements sources and constraints [1]. For example, a product line may address multiple customer groups and thus requirements engineers need to involve representatives of those groups. Support for the elicitation and documentation of common and variable requirements has thus been a focus of past research [34], [35].

[10] http://www.splot-research.org/

The amount of commonality and variability defined in domain requirements engineering has a huge impact on all other product line engineering activities, both in domain and application engineering. A high percentage of common features and common domain requirements in a product line typically require lower effort for designing and realizing the product line. Moreover, common requirements and domain artifacts are essential to engineering a product line platform that is stable yet flexible enough. On the other hand, the extent of variable requirements determines the potential number of different applications that can be derived from the product line and thus has significant impact on whether all goals and needs of the envisioned customers and/or market segments may be satisfied [1]. If a set of differing but related requirements is identified, two principle ways to treat those requirements exist. Those requirements may be defined as variable in the domain requirements. Or, those requirements may be harmonized or generalized and thereby defined as a common domain requirement. Determining how to treat those requirements is clearly a tradeoff decision that has to be made in concert with product management and scoping.

Like in single system development, the requirements of software product lines tend to change frequently. Surprisingly, research in product line evolution has mainly focused on the evolution of variability models, as well as design and realization artifacts, yet has mostly neglected changes in requirements [36], [37], [38].

Open research challenges in **domain requirements engineering** include:

- **Interrelation between scoping and requirements engineering:** How to facilitate continuous interactions between domain requirements engineering and product management? For example, additional common or variable requirements elicited during domain requirements engineering could lead to different scoping decisions. Or, as a result of requirements validation, scoping decisions could be questioned. More generally, how is scoping and requirements engineering aligned to support a smooth definition of an optimal set of common and variable requirements [1], [29]?
- **Interrelation between requirements engineering and other development activities**: In single system development, the definition of the requirements and the definition of the software architecture are tightly intertwined (e.g., see [39]). Requirements serve as the basis for the design of the system architecture. Conversely, findings made during architectural design also influence the definition of the requirements. How to handle the intertwining of requirements engineering and design in software product line engineering?
- **Impact of requirements changes:** How to assess the impact of requirements changes on the commonality and variability of the product line? When and how to evolve domain design and realization artifacts as a result of requirements changes? How to evolve a common requirement into a set of variable ones and how to propagate this change to design, realization, test, etc.? If we are able to handle the evolution of requirements well, we might have an easier job with the evolution of other development artifacts. Can well-documented requirements changes guide and structure the changes for other domain and application artifacts?

4.3 Domain Design

Domain design encompasses all activities for defining the reference architecture of the product line. Numerous SPLE design methods have been advocated in the past [40]. The focus of research has recently shifted from design methods to techniques for modelling variability in the architecture (see Section 3.1).

Traditionally, product line architecture approaches have been *component-based*. In such a setting, variability is realized as component compositions [41] and/or by introducing variation points into the components themselves [42].

More recently, *aspect-oriented architectures* have been proposed to better address cross-cutting features. Cross-cutting features are encapsulated into modular units, the aspects, and composed by means of aspect-oriented mechanisms such as advices, join-points and point-cuts [43]. Traditional aspect-oriented modelling and programming concepts may cause problems during software product line evolution. Anything could be an aspect, and an aspect could address any kind of modification of a model or program. Thus, the traditional aspect-oriented modelling concepts are too generic in a product line setting. To address this problem, researchers have started investigating the use of emerging aspect-oriented mechanisms, such as XPIs [44].

Most recently, *service-oriented architectures* have been considered by the SPLE community. In contrast to a component, which represents a comprehensive piece of software that is part of the software product line, a service represents functionality with associated quality characteristics (typically defined in a service-level agreement) offered by a service provider via a service interface [45]. The service itself or the service provider can change as long as the functionality and the service-level agreement remain the same. Key research results for service-oriented product line architectures include feature-model-based approaches for service variability modelling [46] and approaches for reusing and combining services into service-oriented product line applications [47].

Quality attributes (such as performance, availability, security or safety) have been considered during variability modelling, e.g., using extended feature models [48]. Although variation in quality requirements has been addressed in domain requirements models, quality attributes in domain design models have rarely been considered. An exception is the consideration of performance and availability in domain design [12].

Open research challenges in **domain design** include:

- **Building resilient service-oriented product lines:** How to design service-oriented reference architectures that are resilient to dynamic changes in services provided by third parties [49], [45]? Third party services are typically provided on a contractual basis (e.g., expressed in terms of service level agreements). Even though contract violation may imply penalties for the service provider, this does not guarantee that the functionality and quality will be delivered. As a consequence, product line engineers have only limited control over changes of provisioned services in terms of their functionality or quality, as well as their complete failure or even discontinuation – a key difference to the use of components [45]. How can adaptation mechanisms from single system engineering be adapted to a product line setting? How to build variable architectures that can cope with changes in service availabilities during runtime?
- **Delayed design decision and variability:** Decisions about product line variability, i.e., decisions made by product management, and architectural decisions, i.e., fundamental decisions made during the design of the reference architecture often overlap or influence each other [50]. During domain design, architects may decide to delay design decisions to application engi-

neering. For each delayed design decision they define a variation point and a set of design alternatives (variants). How to manage the interaction between such delayed design decisions and product line variability in the product line architecture?

- **SPLE and cloud computing:** What are the consequences of cloud computing on SPLE? How can SPLE architectures be made cloud-aware? Can we build on research results from single system development to deal with cloud-awareness in an SPLE setting?

- **Variability in quality attributes:** Introducing variability in quality attributes on top of functional variability has a significant impact on the product line architecture, especially since quality attributes are typically the key driver for architectural decisions. How to handle variability in quality attributes during domain design? How to take tradeoff decisions between variability in functionality and in quality attributes when designing the software product line architecture?

4.4 Domain Realization

Domain realization deals with the detailed design and the implementation of the domain artifacts, for example, as reusable components or services. Variability can be realized using the capabilities of existing programming languages, compilers, and linkers [6]. Approaches include the use of inheritance (e.g., implementing alternative sub-classes for an abstract super-class), aspect-oriented programming (e.g., the weaving of alternative code), conditional compilation (e.g., using preprocessor directives such as #ifdef), and binary replacement (e.g., providing the linker with alternative implementations of libraries). Among those approaches, conditional compilation has received significant attention with research outcomes addressing type-safe feature modularity [51] and the treatment of feature dependencies [52].

To explicitly handle feature modularity and feature dependencies (or interactions) at the language level, new types of programming languages have been proposed that consider features and variability as first-class concepts. *Feature-oriented programming (FOP)* is one example. FOP supports the flexible and modular composition of systems from individual features. In FOP, "a feature module encapsulates changes that are made to a program in order to add a new capability or functionality" [53]. In *delta-oriented programming*, a compositional programming language, a product line is realized by a core module and a set of delta modules. The core module implements a valid application developed with single system development techniques. Delta modules specify changes to be applied to the core module to implement additional applications. Changes to the core model include the adding of additional code (as in FOP), but also removing and even the modification of code [54].

The fact that variability often cross-cuts the decomposition structure of the code is a shared concern in domain realization, independent of the programming language used [51], [53]. Cross-cutting variability is addressed by introducing additional composition operations on top of sequential composition [53] or by treating features as aspects [55].

Open research challenges in **domain realization** include:

- **Mapping of product line variability and software variability:** The realization of product line variability often affects more than one code fragment and thus cross-cuts realization artifacts. Conversely, a realization artifact can (in parts) implement more than one product line variability. Thus, the realization of variability typically results in an *m*:*n* mapping of variability and code fragments. How to handle this *m*:*n* mapping? How to support the step-wise refinement of product line variability to software variability? Can scripting languages be extended to manage the step-wise refinement and the *m*:*n* mapping between product line variability and software variability?

4.5 Domain Quality Assurance

Quality assurance of domain artifacts is essential for successful product line engineering. A fault in a domain artifact may affect all applications of the product line in which this artifact is reused. Quality assurance techniques from single-system engineering cannot be directly applied to domain artifacts. As an example, a domain requirements specification can define a variable requirement r, that is related to variant v_1, and a variable requirement $\neg r$ related to variant v_2. Performing a consistency check of the domain requirements specification $R = \{r, \neg r\}$ using quality assurance techniques from single system development would identify a contradiction between r and $\neg r$. Yet, if the variants v_1 and v_2 are defined to be mutually exclusive, the contradicting requirements can never be implemented together in the same application. Thus, the two requirements will never cause an inconsistency. A central challenge for quality assurance techniques in domain engineering is thus the consideration of product line variability [1], [56].

A vast amount of research has focused on strategies and techniques for quality assurance in the presence of variability, including formal verification, static analysis, and (dynamic) testing. The consistency of the variability model is a prerequisite for most domain quality assurance techniques and is established using variability analysis techniques as discussed in Section 3.2.

4.5.1 Formal Verification

Formal verification of product line artifacts has been the focus of numerous research contributions. Prominent verification techniques from single systems engineering have been adapted to the software product line setting [57], including type checking, model checking, and theorem proving. To handle variability during verification, various strategies have been followed, such as checking representative applications, checking features in isolation, or aiming to check all potential applications of the product line [56].

Even though existing product line verification techniques may indicate why they determined an inconsistency, these techniques typically do not provide support for identifying the root cause for a given inconsistency. For example, a model checker can provide an execution trace for an observed violation but does not point out which part of the specification actually led to the invalid trace. Or, a theorem prover can indicate the part of the specification (formula) that could not be proven, while the reason for that may lie elsewhere. It thus can become very challenging for product line engineers to locate and identify a defect in the commonality or variability, especially given the size of product lines in industry.

Moreover, inconsistencies not only occur within an individual artifact model but also between models. Product line verification has started addressing inter-model inconsistencies. However, so far only a limited set of modelling views has been considered, such as use cases, activity diagrams, and scenarios [58].

4.5.2 Domain Testing

As in the development of single systems, the aim of testing in SPLE is to execute the software to uncover the evidence of defects. Research on domain testing has delivered techniques for developing reusable test cases in domain engineering, and reusing

and executing these test cases in application engineering (see Section 5.4). Key results include techniques for defining test cases for different types of tests, including system, integration, and performance tests [59], [60].

In addition, domain testing aims to uncover evidence of defects in domain artifacts before these artifacts are reused in application engineering. Due to the variability defined in the domain artifacts, testing all potential product line applications (i.e., all potential combination of the common and variable artifacts) during domain engineering is impossible [61]. Typical domain testing strategies thus reduce the number of artifact combinations by using pair-wise [62] or *t*-wise testing strategies [63] or by focusing on important features and feature combinations [64].

Open research challenges in **domain quality assurance** include:

- **Causes for failures:** Initial progress has been made to identify the root causes for inconsistencies in variability models [65]. How to extend such approaches to surface root causes for faults and failures in other domain artifacts such as requirements, design artifacts, and code (cf., [66])?
- **Inter-model verification:** Current product line verification techniques mainly focus on single artifact models. How to verify inter-model consistency during domain engineering?
- **Empirical evidence:** Controlled experimentation is an established research methodology to evaluate testing techniques in single system development. With a few exceptions, product line testing research has not yet provided reproducible empirical results [59]. How to establish empirical evidence of the efficiency and effectiveness of domain testing techniques?
- **Empowering additional quality assurance techniques during domain engineering:** So far, research for domain quality assurance has focused on formal verification and testing. Almost no research contributions exist to tailor or extend other successful quality assurance techniques from single software development to the product line setting such as reviews, walkthroughs, or perspective-based reading. This is surprising, since such techniques have proven to be very effective in single system development. So, can reviews, walkthroughs and perspective-based reading techniques be applied in a software product line setting? Is an adaptation of those techniques needed? How to facilitate the validation of variability and commonality and the associated trade-off decisions? Can we gather empirical evidence of the effectiveness of those quality assurance techniques in an SPLE setting?

5. APPLICATION ENGINEERING

5.1 Application Requirements Engineering

During application requirements engineering, the requirements for a specific application are defined. In general, customer- and application-specific requirements should be satisfied by reusing the domain requirements and exploiting the variability defined for the software product line. To this end, the variability is bound to application-specific features (or variants) to satisfy the application-specific requirements. Application variability models have been proposed for documenting the application-specific binding of the product line variability [1], [8].

Many publications in the field convey the impression that a concrete application of the product line can be completely derived from the domain artifacts and thus reduce the application derivation process to a feature selection process. For example, decision models define the decisions to be taken to derive an application of the product line [19]. To this end, a decision model documents the possible decisions, their impacts as well as their ordering and, possibly, their pre-conditions. Variability is indirectly modelled in a decision model through the decisions that need to be taken to resolve variability. To guide users during these decision processes, tools have been suggested [67]. In the extreme, fully automated approaches have been devised that aim at optimal feature selection; e.g., using search-based techniques [68].

In practice, individual applications often cannot be fully realized by reusing domain artifacts alone [8], [9]. Quite often, there are some application-specific requirements that cannot be satisfied by reusing domain requirements and thus have to be realized during application engineering. The handling of application-specific deviations from product line requirements has received very little attention despite its frequent occurrence in practice. A potential solution is to define such application-specific deviations as application-specific variation and document this variation, in addition to the variability bindings, in the application variability model [8].

Open challenges in **application requirements engineering** are:

- **Eliciting application-specific requirements:** Some early research contributions exist that support the elicitation of application-specific requirements. The mere selection of pre-defined features is certainly not a satisfying answer. So how to make use of variability during requirements elicitation? How to enrich or adapt traditional elicitation approaches to leverage product line variability? How to extend decision-model-based approaches to include application-specific deviations from domain requirements?
- **Handling application-specific deviations:** Customer-specific requirements that cannot be fulfilled by the commonality or the variability defined in the product line (i.e., which cannot be mapped to domain artifacts) have to be documented and managed in an appropriate way. How to document application-specific extensions to requirements? How to check their impact on the defined domain requirements? How to map application-specific deviations back to the domain requirements without impeding the efficiency of reusing product line artifacts? How to evaluate and predict the consequences of application-specific requirements deviations early enough? How to evaluate potential alternatives with regard to the domain requirements and the variability of the product line? How to evaluate the impact of application-specific deviations on later application engineering phases like design, testing and maintenance?

5.2 Application Design

Based on the application requirements, an application-specific architecture is derived from the domain architecture. The application architecture is typically a specialization of the reference architecture of the product line [1].

During application design, the design alternatives that have been identified during domain design and that have been documented as variability in the product line platform are evaluated with regard to the application requirements. The alternatives that fit best are chosen accordingly. Yet, in the case of application-specific deviations (see above), additional design decisions may have to be taken in order to derive an architecture that satisfies the application-specific requirements, or, the architecture might even have to be extended or adjusted accordingly. Even an evolution of the product line reference architecture might be triggered.

Open research challenges in **application design** include:

- **Documentation of application design alternatives:** Application-specific design decisions and the chosen design alternatives are documented using the variation points defined in the reference architecture. If application-specific design alternatives caused by application-specific deviations are required, can those also be documented by variation points [69]? If not, should new, application-specific architectural variation points be introduced and related to the application variability model to make the extensions visible and traceable?
- **Impact of application-specific extensions:** How to evaluate potential application-specific architectural adaptations? How to manage such adaptations in application engineering but also during product line evolution? How to evaluate the effect of such adaptation on domain realization and testing? How to choose adaptations with minimal impact?

5.3 Application Realization

During application realization, code artifacts are derived and adjusted based on the application architecture and the application-specific requirements. For example, software configuration techniques are employed to facilitate the parameterization and the composition of the reusable code modules.

Research in application realization delivered techniques for creating consistent configurations of code modules considering various types of configuration files, distributed configuration knowledge [70] and technical configuration constraints [71]. An alternative approach to software configuration is code generation [72]. Code generation techniques for product line applications have mainly adapted techniques from model-driven development and domain-specific languages [16].

Configurable or generative software product lines, a subclass of software product lines, support the derivation of individual applications without programming glue code or modifying the domain components [70]. As mentioned already several times, such an ideal approach is often not possible in practice. In other words, application-specific coding and adjustments are usually required.

Open research challenges in **application realization** include:

- **Framing application-specific programming:** The derivation of application code from reusable software artifacts in a software product line setting is often still a time-consuming and expensive activity in industry, because it usually requires additional, application-specific programming [73]. How can such application-specific programming be framed to avoid undesired side effects? What is a good modular detailed design for application-specific extensions? How to provide programmers with the relevant software product line design information in order to foster product line compliant extensions?
- **Extended configuration mechanism:** Can existing product line configuration approaches be extended to handle application-specific code fragments without changing the underlying domain artifacts? How should such extensions be documented? How to assess the side effects of such extensions on the configurations? Do extensions lead to an inconsistency in a configuration, or may they even neglect a configuration?
- **Product line development environments:** Besides the industrial importance and the very positive experience of applying software product lines in industry, there are almost no product line specific development environments (e.g., IDEs). How to extend commercial IDEs to enable their smooth usage in product line settings? How to support the programmers in making informed trade-off-decisions between application-specific extensions and the principles and guidelines defined for the product line platform during domain engineering?

5.4 Application Quality Assurance

Research for application quality assurance in SPLE has focused on application testing. In general, product line testing techniques advocate the early testing during domain engineering (see Section 4.5). Domain testing can uncover the evidence for critical faults, such that these faults can be corrected before they affect several applications of the product line [61]. However, domain testing is not sufficient. Due to the variability of the reusable artifacts, it is impossible – except for trivial product lines – to comprehensively test all potential applications during domain engineering. Moreover, if specific variants are developed based on concrete customer demands (e.g., see the discussion in Section 4.2) such variants and their potential side effects can only be tested during application engineering.

Research on application testing has mainly delivered techniques for deriving test cases from reusable test artifacts developed in domain engineering [59], [60]. Some application testing techniques aim at testing application-specific adaptations of domain artifacts. In addition, they aim to minimize the retesting of parts that have already been tested for another application of the product line, thereby representing a special case of regression testing [74], [59]. However, those techniques do not consider what has already been tested during domain engineering.

Open research challenges in **application testing** include:

- **Minimizing test redundancy:** Test artifacts and results obtained in domain engineering should be considered in application testing to avoid the replication of test executions which are not really required. However, systematic approaches for considering test results from domain engineering during application engineering are rare. Extensions for testing techniques which avoid unnecessary redundancy between domain and application engineering tests are still missing. Similar, how to consider tests already executed for other applications during application engineering? How to formally assess side effects of application-specific bindings and extensions to determine unrequired as well as required test re-executions for a given application?
- **Correct variability bindings:** The variants bound for a specific application have to conform to the definition in the application variability model. There are several reasons for checking the correct binding of the variability. For example, correct binding avoids delivering features in an application the customer hasn't paid for. Thus, evidence should be established that the derived application includes all features defined in the application variability model, but not more [13], [59]. Or, the incorporation of non-required features may increase the vulnerability for attacks. For security reasons, there should not be any non-required features (and thus non-required code) in a product line application. So, how to ensure that only selected variants are bound in an application? And, how to formally prove that the variability bindings are correct? How to consider the different binding times during such proves, e.g., during development, compilation, linking and run-time?
- **Empowering additional quality assurance techniques:** So far, research for application quality assurance has mainly focused on testing. As for domain engineering, in application engineering

almost no research contribution exist which lifts other quality assurance techniques from single software development to application engineering. How to deliver effective and efficient application quality assurance techniques, such as reviews, walkthroughs or perspective-based reading techniques, while making use of domain quality assurance results? How to consider application-specific variability bindings as well as application-specific extensions during application quality assurance?

- **Empirical evidence:** Empirical results on efficiency and effectiveness of application testing techniques and other quality assurance techniques are missing. Thus, the challenges sketched for domain quality assurance techniques in Section 4.5 also apply for application quality assurance techniques.

6. EMERGING RESEARCH CHALLENGES

SPLE research has produced very impressive results over the past seven years. Some research areas, like variability modelling and formal verification, have attracted many researchers to tackle actual challenges. Other research areas of high practical relevance, like product line quality assurance techniques, scoping, domain design, application requirements engineering, or application design and realization have largely been neglected. In the previous sections we summarized the major research achievements and sketched open research issues and challenges in variability modeling and analysis, domain engineering and application engineering.

In addition to those challenges, major trends in software engineering and technology lead to new research challenges. We elaborate on those challenges in this section.

6.1 Variability Management in Non-Product-Line Settings

In contrast to "opportunistic" and "ad-hoc" software reuse approaches, SPLE follows a proactive, strategy-driven reuse approach for software artifacts. Reuse is planned at all levels of the organization and throughout all phases of domain and application engineering. However, there are cases in which a strategic and planned definition of a product line is not economically viable or not even possible. Beyond the investment in technical design and development of domain artifacts, the introduction of SPLE usually requires a change in the processes and the organization structure (see Section **Fehler! Verweisquelle konnte nicht gefunden werden.**). This typically requires significant investments. Creating convincing business cases which demonstrate that the introduction of a SPLE approach will lead to the expected return-on-investment is often not easy, especially since lower maintenance costs are hard to predict. Therefore, and for many other reasons, instead of following an SPLE approach, software systems are quite often created by "cloning" existing ones, i.e., by copying and modifying requirements, architecture and code of preceding systems. We strongly believe that in the future, the number of cases in which the "copy-and-modify" (aka. "clone-and-own") approach is used will even increase. Among others, reasons for this are more frequent demand changes, the need to adapt the applications to new technology and service offerings at run-time, or the rapid changes of the system context and the system requirements. All of those will make a prediction of the scope of a potential product line much harder if not impossible.

So, how to systematically manage the variability of related, single applications in a non-product line setting? How to systematically identify and manage the commonalities, variability and application-specific artifacts? The principles of product line variability management can also be applied in such settings and can improve current industrial practice significantly. For example, software configuration management tools may be extended with explicit variability management facilities (e.g., see [75]). So, how to facilitate the identification and management of variability in a non-product line setting? For instance, how to derive variability information based on "copy-and-modify" activities executed by the engineers?

6.2 Leveraging Instantaneous Feedback

Cloud computing aims to provide seamless adaptations of the infrastructure in real-time and facilitates measuring infrastructure usage and system execution parameters in real-time. When combined with the Internet of Things [76], system execution data can be enriched with data about the system context obtained by thousands of sensors. Big data analytics facilitates turning all this data into potential actionable insights with very low latency.

Together these emerging technologies empower software developers and operators to continuously adjust the system based on instantaneous feedback obtained from system execution and the system context [77]. As a consequence, the tension between upfront investment and planning of a software product line and the increased agility fostered by instantaneous feedback and continuous deployment must be reconciled.

Early experience indicates that for a reactive product line setting combining agile techniques and SPLE works well [78] and that agile principles can be applied during application engineering [79]. How to leverage this experience in order to master the dynamicity resulting from big data analytics and cloud technology? Can SPLE principles be applied in such highly dynamic settings? Can changes of product line variability and commonalities be automatically inferred from analyzing operational and contextual data? And if so, how to adjust the overall product line engineering setting to master such highly frequent changes?

6.3 Open World Assumption

Driven by the Internet of Services, the Internet of Things, and the emergence of new highly distributed systems, such as cyber-physical systems and ultra-large-scale systems [80], the need for software to live in an *open* and highly dynamic world is becoming mandatory. Traditionally, software development was based on the closed world assumption, which means that the boundary between the system and its context is known during design-time and that the context does not change while the system is executing [81]. In contrast, open world systems cannot be specified completely during design-time due to incomplete knowledge about, for instance, services and their actual quality provided during run-time, sensors available during system operation to obtain environment information, the availability of other systems to interact and cooperate with, as well as the amount and quality of data obtained. The development of future systems thus has to live with uncertainty in the specifications. During operation, such systems must frequently adapt to the dynamic changes faced during run-time [82], [45].

Dynamic software product lines (DSPLs) aim to address context changes during system execution by postponing variability binding to run-time. An application is empowered to dynamically reconfigure itself by choosing at run-time an appropriate binding predefined in the product line variability. Research in dynamic software product lines has focused on employing variability models to define the configuration space of a product line application, thereby describing possible and permissible run-time adaptations [83]. DSPLs are applicable if future context conditions are known

during design-time [84], [85]. Unfortunately, for cyber-physical and ultra-large-scale systems foreseeing future context conditions and defining appropriate adaptation options during design-time is often not possible.

For addressing the open world assumption in SPLE major research challenges have to be mastered, including:

- **Exploration of autonomic computing principles:** Autonomic computing provides fundamental models, algorithms and techniques to adjust systems to known and unknown situations during their execution. Can such approaches be used in a software product line setting? What kinds of adaptations are required to make them fit for an SPLE setting? For example, learning and reasoning techniques are applied in control loops of adaptive systems to deal with unknown situations (e.g., [86], [87]). Can such principles be applied in SPLE settings, e.g., to update variability models and product line artifacts to unknown situations in a similar way? Or, can improved evolution techniques for software ecosystems handle such situations [88]?

- **Reasoning in the presence of variability and uncertainty:** To deal with gaps in the specification and architecture during design-time, formal reasoning mechanisms and the underlying models have to be extended to deal with uncertainty and variability at the same time. Such extended modelling and reasoning approaches could certainly contribute partial solutions for well-defined cases, but will hardly scale to solve all the problems associated with the openness of cyber-physical and ultra-large-scale systems.

- **Human-in-the-loop adaptations:** In cockpits and control towers, humans (e.g., the operators) are involved to interpret data, to judge the criticality of a given situation and to decide during run-time about the adaption of an application as a reaction to foreseeable and unforeseeable changes and exceptions. Can we learn from such principles to develop solutions which can adjust product line applications during run-time to unforeseeable situations? Do such principles provide a path for developing system adaptations in which the closed world assumption does not hold anymore?

- **Run-time quality assurance:** Adaptations, especially in an open setting, are prone to specific failures that do not occur in closed settings. For example, run-time decisions and adaptations may lead to conflicting, faulty or even inconsistent configurations. To address (context) situations unknown during design-time, quality can only be assured partially and under certain assumptions. How to model such assumptions? How to check if the assumptions hold in the actual situation? Or, even more generally, how to ensure the quality of an application during run-time if not all potential adaptations of the application are known and predefined? Can run-time quality assurance techniques from service engineering be adjusted to such settings?

7. CONCLUSIONS

We surveyed over 600 relevant papers published in established conferences and journals over the past seven years. Overall, the research progress achieved is impressive! For example, significant contributions have been established in the areas of variability modelling and in formally verifying product line artifacts. Even so, many research challenges remain. We summarized the research achievements and the open challenges using a standardized framework for software product line engineering.

Many of the open research challenges have existed for already quite some time and are highly relevant for industry. It is thus a bit surprising that they have not attracted more research efforts. Examples of such challenges include product line quality assurance techniques, scoping, domain design, application requirements engineering, as well as application design and realization.

In addition, we highlighted three trends that will have an impact on SPLE research in the next decade: (1) managing variability in non-product-line settings, (2) leveraging instantaneous feedback from big data and cloud computing during SPLE, (3) addressing the open world assumption in software product line settings. Those trends clearly indicate that research opportunities arise at the intersection between software product line engineering and service-oriented computing, cloud computing, big data analytics, autonomic computing and adaptive systems, to name the most important ones. This in turn requires closer cooperation between the currently often separated research communities. We should thus stimulate multi-disciplinary forums (such as workshops and conferences) and joint research projects as key instruments for fostering the exchange and the cooperation between the different research communities.

8. ACKNOWLEDGMENTS

Reflecting on the achievements and open research challenges in the very active field of software product line engineering and variability management has been an inspiring endeavor. This paper greatly benefited from the very constructive and valuable feedback received from David Garlan, Linda Northrop, Klaus Schmid and Frank van der Linden on an earlier version of the paper. Thanks a million to all of you! We also like to thank the members of our research group for the support provided, especially André Heuer, Richard Pohl and Vanessa Stricker.

This work has been partially supported by the DFG (German Research Foundation) under grant PO 607/4-1 (KOPI).

9. REFERENCES

[1] K. Pohl, G. Böckle and F. van der Linden, Software Product Line Engineering: Foundations, Principles, and Techniques, Berlin, Heidelberg, New York: Springer, 2005.

[2] P. Clements and L. Northrop, Software Product Lines: Practices and Patterns, Reading, USA: Addison-Wesley, 2001.

[3] J. Coplien, D. Hoffmann and D. Weiss, "Commonality and Variability in Software Engineering," *IEEE Software,* vol. 15, no. 6, pp. 37-45, 1998.

[4] A. Metzger, P. Heymans, K. Pohl, P.-Y. Schobbens and G. Saval, "Disambiguating the Documentation of Variability in Software Product Lines: A Separation of Concerns, Formalization and Automated Analysis," in *15th Int'l Requirements Engineering Conference (RE 2007)*, New Delhi, India, 2007.

[5] F. van der Linden, K. Schmid and E. Rommes, Software Product Lines in Action, Berlin, Heidelberg, New York: Springer, 2007.

[6] R. Capilla, J. Bosch and K.-C. Kang, Systems and Software Variability Management, Heidelberg, New York: Springer, 2013.

[7] K. Schmid and M. Verlage, "The Economic Impact of

Product Line Adoption and Evolution," *IEEE Software,* vol. 19, no. 6, pp. 50-57, 2002.

[8] G. Halmans, K. Pohl and E. Sikora, "Documenting Application-Specific Adaptations in Software Product Line Engineering," in *20th Int'l Conference on Advanced Information Systems Engineering (CAiSE 2008)*, Montpellier, France, 2008.

[9] S. Adam and K. Schmid, "Effective Requirements Elicitation in Product Line Application Engineering: An Experiment," in *19th Int'l Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2013)*, Essen, Germany, 2013.

[10] K. Schmid and E. Santana de Almeida, "Product Line Engineering," *IEEE Software,* vol. 30, no. 4, pp. 24-30, 2013.

[11] M. Svahnberg, J. van Gurp and J. Bosch, "A Taxonomy of Variability Realization Techniques," *Software: Practice and Experience,* vol. 35, no. 8, pp. 705-754, 2005.

[12] M. Galster, D. Weyns, D. Tofan, B. Michalek and P. Avgeriou, "Variability in Software Systems: A Systematic Literature Review," *IEEE Trans. Softw. Eng.,* available online 2013.

[13] A. Reuys, S. Reis, E. Kamsties and K. Pohl, "The ScenTED Method for Testing Software Product Lines," in *Software Product Lines: Research Issues in Engineering and Management*, Berlin, Heidelberg, Springer, 2006, pp. 479-520.

[14] H. Heuer, V. Stricker, C. Budnik, S. Konrad, K. Lauenroth and K. Pohl, "Defining Variability in Activity Diagrams and Petri Nets," *Sci. Comput. Program.,* vol. 78, no. 12, pp. 2414-2432, 2013.

[15] P. Shaker, J. Atlee and S. Wang, "A Feature-oriented Requirements Modelling Language," in *20th Int'l Requirements Engineering Conference (RE 2012)*, Chicago, USA, 2012.

[16] M. Völter and E. Visser, "Product Line Engineering Using Domain-Specific Languages," in *15th Int'l Software Product Line Conference (SPLC 2011)*, Munich, Germany, 2011.

[17] D. Benavides, S. Segura and A. Ruiz-Cortés, "Automated Analysis of Feature Models 20 Years Later: A Literature Review," *Inform. Sys.,* vol. 35, no. 6, pp. 615-636, 2010.

[18] H. Eichelberger, C. Kröher and K. Schmid, "An Analysis of Variability Modeling Concepts: Expressiveness vs. Analyzability," in *13th Int'l Conference on Software Reuse (ICSR 2013)*, Pisa, Italy, 2013.

[19] D. Dhungana, P. Grünbacher and R. Rabiser, "The DOPLER Meta-Tool for Decision-oriented Variability Modeling: A Multiple Case Study," *Autom. Softw. Eng.,* vol. 18, no. 1, pp. 77-114, 2011.

[20] F. Heidenreich, P. Sanchez, J. Santos and others, "Relating Feature Models to Other Models of a Software Product Line," in *Transactions on Aspect-Oriented Software Development VII*, Heidelberg, Springer, 2010, pp. 69-114.

[21] A. Classen, M. Cordy, P.-Y. Schobbens, P. Heymans, A. Legay and J.-F. Raskin, "Featured Transition Systems: Foundations for Verifying Variability-Intensive Systems and Their Application to LTL Model Checking," *IEEE Trans. Softw. Eng.,* vol. 39, no. 8, pp. 1069-1089, 2013.

[22] T. Berger, R. Rublack, D. Nair, J. Atlee, M. Becker, K. Czarnecki and A. Wasowski, "A Survey of Variability Modeling in Industrial Practice," in *7th Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2013)*, Pisa, Italy, 2013.

[23] H. Eichelberger and K. Schmid, "A Systematic Analysis of Textual Variability Modeling Languages," in *17th Int'l Software Product Line Conference (SPLC 2013)*, Tokyo, Japan, 2013.

[24] M. Bittner, M.-O. Reiser and M. Weber, "A Case Study on Tool-Supported Multi-level Requirements Management in Complex Product Families," in *16th Int'l Working Conference Requirements Engineering: Foundation for Software Quality (REfSQ 2010)*, Essen, Germany, 2010.

[25] D. Benavides, A. Felfernig, J. Galindo and F. Reinfrank, "Automated Analysis in Feature Modelling and Product Configuration," in *13th Int'l Conference on Software Reuse (ICSR 2013)*, Pisa, Italy, 2013.

[26] R. Pohl, V. Stricker and K. Pohl, "Measuring the Structural Complexity of Feature Models," in *28th Int'l Conference on Automated Software Engineering (ASE 2013)*, Palo Alto, USA, 2013.

[27] A. Helferich, K. Schmid and G. Herzwurm, "Product Management for Software Product Lines: An Unsolved Problem?," *Commun. ACM,* vol. 49, no. 12, pp. 66-67, 2006.

[28] I. John and M. Eisenbarth, "A Decade of Scoping: A Survey," in *13th Int'l Software Product Line Conference (SPLC 2009)*, San Francisco, USA, 2009.

[29] J. Gillain, S. Faulkner, P. Heymans, I. Jureta and M. Snoeck, "Product Portfolio Scope Optimization based on Features and Goals," in *16th Int'l Software Product Line Conference (SPLC 2012)*, Salvador, Brazil, 2012.

[30] R. van Ommering and J. Bosch, "Widening the Scope of Software Product Lines: From Variation to Composition," in *2nd Int'l Software Product Line Conference (SPLC)*, San Diego, USA, 2002.

[31] S. Bühne, K. Lauenroth, K. Pohl and M. Weber, "Modelling Features for Multi-Criteria Product-Lines in the Automotive Industry," in *ICSE Workshop on Software Engineering for Automotive Systems (SEAS 2004)*, Edinburgh, UK, 2004.

[32] G. Holl, P. Grünbacher and R. Rabiser, "A Systematic Review and an Expert Survey on Capabilities Supporting Multi Product Lines," *Information and Software Technology,* vol. 54, no. 8, pp. 828-852, 2012.

[33] J. Bosch and P. Bosch-Sijtsema, "From Integration to Composition: On the Impact of Software Product Lines, Global Development and Ecosystems," *Journal of Systems and Software,* vol. 83, no. 1, pp. 67-76, 2010.

[34] N. Niu and S. Easterbrook, "Extracting and Modeling Product Line Functional Requirements," in *16th Int'l Requirements Engineering Conference (RE 2008)*, Barcelona, Spain, 2008.

[35] E. Bagheri, F. Ensan and D. Gasevic, "Decision Support for

the Software Product Line Domain Engineering Lifecycle," *Automated Software Engineering,* vol. 19, no. 3, pp. 335-377, 2012.

[36] C. Seidl, F. Heidenreich and U. Aßmann, "Co-evolution of Models and Feature Mapping in Software Product Lines," in *16th Int'l Software Product Line Conference (SPLC 2012)*, Salvador, Brazil, 2012.

[37] L. Neves, L. Teixeira, D. Sena, V. Alves, U. Kulezsa and P. Borba, "Investigating the Safe Evolution of Software Product Lines," in *10th Int'l Conference on Generative Progr. and Component Eng. (GPCE 2011)*, Portland, USA, 2011.

[38] X. Peng, Y. Yu and W. Zhao, "Analyzing Evolution of Variability in a Software Product Line: From Contexts and Requirements to Features," *Information and Software Technology,* vol. 53, no. 7, pp. 707-721, 2011.

[39] K. Pohl, Requirements Engineering: Fundamentals, Principles, and Techniques, Heidelberg: Springer, 2010.

[40] M. Matinlassi, "Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA," in *26th Int'l Conference on Software Engineering (ICSE 2004)*, Edinburgh, UK, 2004.

[41] M. Janota and G. Botterweck, "Formal Approach to Integrating Feature and Architecture Models," in *11th Int'l Conference on Fundamental Approaches to Software Engineering (FASE 2008)*, Budapest, Hungary, 2008.

[42] A. Haber, H. Rendel, B. Rumpe, I. Schaefer and F. van der Linden, "Hierarchical Variability Modeling for Software Architectures," in *15th Int'l Software Product Line Conference (SPLC 2011)*, Munich, Germany, 2011.

[43] E. Figueiredo, N. Cacho, C. Sant'Anna and others, "Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability," in *30th Int'l Conference on Software Engineering (ICSE 2008)*, Leipzig, Germany, 2008.

[44] M. Dias, L. Tizzei, C. F. Rubira, A. Garcia and J. Lee, "Leveraging Aspect-Connectors to Improve Stability of Product-Line Variabilities," in *4th Int'l Workshop on Variability Modelling of Software-Intensive Systems (VaMoS 2010)*, Linz, Austria, 2010.

[45] E. Di Nitto, C. Ghezzi, A. Metzger, M. P. Papazoglou and K. Pohl, "A Journey to Highly Dynamic, Self-adaptive Service-based Applications," *Autom. Softw. Eng.,* vol. 15, no. 3-4, pp. 313-341, 2008.

[46] B. Mohabbati, M. Asadi, D. Gasevic, M. Hatala and H. Müller, "Combining Service-orientation and Software Product Line Engineering: A Systematic Mapping Study," *Information and Software Technology,* vol. 55, no. 11, pp. 1845-1859, 2013.

[47] J. Lee, D. Muthig and M. Naab, "A Feature-oriented Approach for Developing Reusable Product Line Assets of Service-based Systems," *Journal of Systems and Software,* vol. 83, no. 7, pp. 1123-1136, 2010.

[48] V. Myllärniemi, M. Raatikainen and T. Männistö, "A Systematically Conducted Literature Review: Quality Attribute Variability in Software Product Lines," in *16th Int'l Software Product Line Conference (SPC 2012)*, Salvador, Brazil, 2012.

[49] J. Camara, R. de Lemos, C. Ghezzi and A. Lopes, Assurances for Self-Adaptive Systems, Heidelberg: Springer, 2013.

[50] I. Lytra, H. Eichelberger, H. Tran, G. Leyh, K. Schmid and U. Zdun, "On the Interdependence and Integration of Variability and Architectural Decisions," in *8th Int'l Workshop on Variability Modelling of Software-intensive Systems (VaMoS 2014)*, Sophia Antipolis, France, 2014.

[51] C. Kästner, K. Ostermann and S. Erdweg, "A Variability-aware Module System," in *Int'l Conference on Object Oriented Programming Systems Languages and Applications (OOPSLA 2012)*, Tucson, USA, 2012.

[52] M. Ribeiro, F. Queiroz, P. Borba, T. Toledo, C. Brabrand and S. Soares, "On the Impact of Feature Dependencies when Maintaining Preprocessor-based Software Product Lines," in *10th Int'l Conference on Generative Progr. and Component Eng. (GPCE 2012)*, Portland, USA, 2012.

[53] D. Batory, P. Höfner and J. Kim, "Feature Interactions, Products, and Composition," in *10th Int'l Conference on Generative Progr. and Component Eng. (GPCE 2011)*, Portland, USA, 2011.

[54] A. Haber, K. Hölldobler, C. Kolassa, M. Look, B. Rumpe, K. Müller and I. Schaefer, "Engineering Delta Modeling Languages," in *17th Int'l Software Product Line Conference (SPLC 2013)*, Tokyo, Japan, 2013.

[55] H. Cho, K. Lee and K. Kang, "Feature Relation and Dependency Management: An Aspect-Oriented Approach," in *12th Int'l Software Product Line Conference (SPLC 2008)*, Limerick, Ireland, 2008.

[56] K. Lauenroth, A. Metzger and K. Pohl, "Quality Assurance in the Presence of Variability," in *Intentional Perspectives on Information Systems Engineering*, Heidelberg, Springer, 2010, pp. 319-334.

[57] S. Apel, A. von Rhein, P. Wendler, A. Größlinger and D. Beyer, "Strategies for product-line verification: case studies and experiments," in *35th Int'l Conference on Software Engineering (ICSE '13)*, San Francisco, USA, 2013.

[58] J. Greenyer, A. Molzam Sharifloo, M. Cordy and P. Heymans, "Features Meet Scenarios: Modeling and Consistency-Checking Scenario-based Product Line Specifications," *Requirements Engineering Journal,* vol. 18, no. 2, pp. 175-198, 2013.

[59] J. Lee, S. Kang and D. Lee, "A Survey on Software Product Line Testing," in *16th Int'l Software Product Line Conference (SPLC 2012)*, Salvador, Brazil, 2012.

[60] E. Engström and P. Runeson, "Software Product Line Testing: A Systematic Mapping Study," *Information and Software Technology,* vol. 53, no. 1, pp. 2-13, 2011.

[61] K. Pohl and A. Metzger, "Software Product Line Testing," *Commun. ACM,* vol. 49, no. 12, pp. 78-81, 2006.

[62] M. Cohen, M. B. Dwyer and J. Shi, "Constructing Interaction Test Suites for Highly-Configurable Systems in the Presence of Constraints: A Greedy Approach.," *IEEE Trans. Soft. Eng.,* vol. 34, no. 5, pp. 633-650, 5 34 2008.

[63] G. Perrouin, S. Oster, S. Sen, J. Klein, B. Baudry and Y. Le

Traon, "Pairwise Testing for Software Product Lines: Comparison of Two Approaches," *Software Quality Journal,* vol. 20, no. (3-4), pp. 605-643, 2012.

[64] M. F. Johansen, Ø. Haugen, F. Fleurey, A. G. Eldegard and T. Syversen, "Generating Better Partial Covering Arrays by Modeling Weights on Sub-product Lines," in *15th Int'l Conference on Model Driven Engineering Languages and Systems (MODELS 2012)*, Innsbruck, Austria, 2012.

[65] J. White, D. Benavides, D. Schmidt, P. Trinidad, B. Dougherty and A. Ruiz-Cortes, "Automated Diagnosis of Feature Model Configurations," *Journal of Systems and Software,* vol. 83, no. 7, pp. 1094-1107, 2010.

[66] R. Lopez-Herrejon and A. Egyed, "Towards Fixing Inconsistencies in Models with Variability," in *6th Int'l Workshop on Variability Modeling of Software-Intensive Systems (VaMoS 2012)*, Leipzig, Germany, 2012.

[67] R. Rabiser, P. Grünbacher and M. Lehofer, "A Qualitative Study on User Guidance Capabilities in Product Configuration Tools," in *27th Int'l Conference on Automated Software Engineering (ASE 2012)*, Essen, Germany, 2012.

[68] A. Sayyad, T. Menzies and H. Ammar, "On the Value of User Preferences in Search-based Software Engineering: A Case Study in Software Product Lines," in *35th Int'l Conference on Software Engineering (ICSE 2013)*, San Francisco, USA, 2013.

[69] M. Galster, P. Avgeriou, D. Weyns and T. Männistö, "Variability in Software Architecture: Current Practice and Challenges," *ACM SIGSOFT Software Engineering Notes,* vol. 36, no. 5, pp. 30-32, 2011.

[70] E. Cirilo, U. Kulesza, A. Garcia, D. Cowan, P. Alencar and C. Lucena, "Configurable Software Product Lines: Supporting Heterogeneous Configuration Knowledge," in *13th Int'l Conference on Software Reuse (ICSR 2013)*, Pisa, Italy, 2013.

[71] C. Elsner, P. Ulbrich, D. Lohmann and W. Schröder-Preikschat, "Consistent Product Line Configuration across File Type and Product Line Boundaries," in *14th Int'l Software Product Line Conference (SPLC 2010)*, Jeju Island, South Korea, 2010.

[72] G. Perrouin, J. Klein, N. Guelfi and J.-M. Jezequel, "Reconciling Automation and Flexibility in Product Derivation," in *12th Int'l Software Product Line Conference (SPLC 2008)*, Limerick, Ireland, 2008.

[73] R. Rabiser, P. O'Leary and I. Richardson, "Key Activities for Product Derivation in Software Product Lines," *Journal of Systems and Software,* vol. 84, no. 2, pp. 285-300, 2011.

[74] V. Stricker, A. Metzger and K. Pohl, "Avoiding Redundant Testing in Application Engineering," in *14th Int'l Software Product Line Conference (SPLC 2010)*, Jeju Island, South Korea, 2010.

[75] J. Rubin, A. Kirshin, G. Botterweck and M. Chechik, "Managing Forked Product Variants," in *16th Int'l Software Product Line Conference (SPLC 2012)*, Salvador, Brazil, 2012.

[76] L. Atzori, A. Iera and G. Morabito, "The Internet of Things: A Survey," *Computer Networks,* vol. 54, no. 15, pp. 2787-2805, 2010.

[77] J. Bosch, "Building Products as Innovation Experiment Systems," in *3rd Int'l Conference on Software Business (ICSOB 2012)*, Cambridge, USA, 2012.

[78] K. Cooper and X. Franch, "Editorial," *Journal of Systems and Software,* vol. 81, no. 6, pp. 841-842, 2008.

[79] J. Díaz, J. Pérez, P. P. Alarcón and J. Garbajosa, "Agile Product Line Engineering: A Systematic Literature Review," *Software: Practice and Experience,* vol. 41, no. 8, pp. 921-941, 2011.

[80] L. Northrop, "Does Scale Really Matter? Ultra-large-scale Systems Seven Years After the Study (Keynote)," in *35th Int'l Conference on Software Engineering (ICSE 2013)*, San Francisco, USA, 2013.

[81] L. Baresi, E. Di Nitto and C. Ghezzi, "Toward Open-World Software: Issues and Challenges," *Computer,* vol. 39, no. 10, pp. 36-43, 2006.

[82] J. Kramer and J. Magee, "Self-Managed Systems: an Architectural Challenge," in *ICSE 2007 Workshop on the Future of Software Engineering (FOSE 2007)*, Minneapolis, USA, 2007.

[83] M. Hinchey, S. Park and K. Schmid, "Building Dynamic Software Product Lines," *IEEE Computer,* vol. 45, no. 10, pp. 22-26, 2012.

[84] P. Sawyer, N. Bencomo, J. Whittle, E. Letier and A. Finkelstein, "Requirements-Aware Systems: A Research Agenda for RE for Self-adaptive Systems," in *18th Int'l Requirements Engineering Conference (RE 2010)*, Sydney, Australia, 2010.

[85] P. Sawyer, R. Mazo, D. Diaz, C. Salinesi and D. Hughes, "Using Constraint Programming to Manage Configurations in Self-Adaptive Systems," *IEEE Computer,* vol. 45, no. 10, pp. 56-63, 2012.

[86] D. Sykes, D. Corapi, J. Magee, J. Kramer, A. Russo and K. Inoue, "Learning Revised Models for Planning in Adaptive Systems," in *35th Int'l Conference on Software Engineering (ICSE 2013)*, San Francisco, USA, 2013.

[87] G. Perrouin, B. Morin, F. Chauvel, F. Fleurey, J. Klein, Y. Le-Traon, O. Barais and J.-M. Jezequel, "Towards Flexible Evolution of Dynamically Adaptive Systems," in *34th Int'l Conference on Software Engineering (ICSE 2012)*, Zurich, Switzerland, 2012.

[88] R. Capilla, J. Bosch, P. Trinidad, A. Ruiz-Cortés and M. Hinchey, "An Overview of Dynamic Software Product Line Architectures and Techniques," *Journal of Systems and Software,* vol. 91, pp. 3-23, May 2014.

[89] J. Rubin and M. Chechik, "Combining Related Products into Product Lines," in *15th Int'l Conference on Fundamental Approaches to Software Engineering (FASE 2012)*, Tallinn, Estonia, 2012.

[90] S. She, R. Lotufo, T. Berger, A. Wasowski and K. Czarnecki, "Reverse Engineering Feature Models," in *33rd Int'l Conference on Software Engineering (ICSE 2011)*, Waikiki, USA, 2011.