HAL
open science

The state of adoption and the challenges of systematic variability management in industry

Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, Jabier Martinez

To cite this version:

Thorsten Berger, Jan-Philipp Steghöfer, Tewfik Ziadi, Jacques Robin, Jabier Martinez. The state of adoption and the challenges of systematic variability management in industry. Empirical Software Engineering, 2020, 25 (3), pp.1755-1797. 10.1007/s10664-019-09787-6 . hal-03968100

HAL Id: hal-03968100

https://paris1.hal.science/hal-03968100

Submitted on 1 Feb 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Empirical Software Engineering (2020) 25:1755–1797
https://doi.org/10.1007/s10664-019-09787-6

The state of adoption and the challenges of systematic variability management in industry

Thorsten Berger[1] · Jan-Philipp Steghöfer[1] · Tewfik Ziadi[2] · Jacques Robin[3] · Jabier Martinez[4]

Published online: 4 April 2020
© The Author(s) 2020

Abstract
Handling large-scale software variability is still a challenge for many organizations. After decades of research on variability management concepts, many industrial organizations have introduced techniques known from research, but still lament that pure textbook approaches are not applicable or efficient. For instance, software product line engineering—an approach to systematically develop portfolios of products—is difficult to adopt given the high upfront investments; and even when adopted, organizations are challenged by evolving their complex product lines. Consequently, the research community now mainly focuses on re-engineering and evolution techniques for product lines; yet, understanding the current state of adoption and the industrial challenges for organizations is necessary to conceive effective techniques. In this multiple-case study, we analyze the current adoption of variability management techniques in twelve medium- to large-scale industrial cases in domains such as automotive, aerospace or railway systems. We identify the current state of variability management, emphasizing the techniques and concepts they adopted. We elicit the needs and challenges expressed for these cases, triangulated with results from a literature review. We believe our results help to understand the current state of adoption and shed light on gaps to address in industrial practice.

Keywords Variability management · Software product lines · Multiple-case study · Challenges

1 Introduction

Companies often need to engineer a portfolio of software variants instead of one-size-fits-all solutions. Creating variants allows tailoring systems towards varying stakeholder

Jabier Martinez contributed to this work during his PostDoc position at the Sorbonne University, France

Communicated by: Tao Yue

✉ Thorsten Berger
thorsten.berger@chalmers.se

Extended author information available on the last page of the article.

requirements—different functionalities, but also non-functional requirements, such as performance or power consumption. Variant-rich systems are especially common in traditional engineering domains including automotive, industrial automation, and telecommunication. In addition, recent trends, such as the Internet of Things (IoT) (Atzori et al. 2010), cyber-physical systems (Krüger et al. 2017; Romero et al. 2015) or robotics (Garcia et al. 2019), further increase the need for customization.

Software product line engineering (SPLE) aims at effectively engineering a variant-rich system—a software product line—in an application domain. SPLE advocates establishing an integrated software platform from which individual variants can be derived, typically in an automated, configuration-driven process. SPLE provides a range of dedicated concepts, including processes, modeling techniques or design patterns, supported by commercial and open-source tools, such as pure::variants (Beuche 2004), Gears (Krueger 2007) or FeatureIDE (Kästner et al. 2009).

Recognizing the benefits, especially the radically decreased time-to-market for new variants, industry has adopted SPLE concepts (Jepsen and Beuche 2009; Flores et al. 2012; Berger et al. 2013a; Mohagheghi and Conradi 2007; Chen and Ali Babar 2010; Bastos et al. 2017; Thörn 2010) at different levels of maturity (Bosch 2002). Yet, many organizations still lament an adoption barrier. In fact, most organizations create variants in ad hoc ways (Berger et al. 2013a), such as clone & own, which is simple and cheap (Dubinsky et al. 2013; Businge et al. 2018; Stanciulescu et al. 2015; Staples and Hill 2004), but does not scale with the number of variants and then requires product-line migration efforts (Assunção et al. 2017; Faust and Verhoef 2003a; Jepsen et al. 2007b; Debbiche et al. 2019; Akesson et al. 2019; Fenske et al. 2014). When adopted, a product line can also limit flexibility, since evolving the platform affects many variants (Melo et al. 2016). To improve this situation, we need to improve our empirical understanding of the state of product-line adoption and the needs for improvement in industrial practice.

Such an updated empirical understanding of industrial needs helps steering the scope of research efforts. Consider the many SPLE concepts that have been conceived by researchers. Already until 2011, a survey identified 91 variability management approaches (Chen and Babar 2011), most of which rely on feature modeling to specify variability information (Kang et al. 1990; Czarnecki et al. 2012; Acher et al. 2013a; Nesic et al. 2019). Another trend was to build hundreds of dedicated analyses for product lines, typically by lifting single-system analyses (e.g., model checking) (Midtgaard et al. 2014; Liebig et al. 2013; Sattler et al. 2018). A recent survey (Thüm et al. 2014) identified 123 analyses from the literature, including lifted type-checking, static-analysis, and model-checking techniques. Already eight years ago, a survey (Benavides et al. 2010) identified 30 different feature-model analyses in 53 papers. Other analyses check the consistency between feature models and implementation artifacts or analyze code properties. Notably, more recent work (Mukelabai et al. 2018b) shows that industrial needs substantially deviate from the state of the art, while most analyses are not applicable in industrial contexts. Likewise, traceability of features across a product line's lifecycle is another challenge (Vale et al. 2017), where the state of the art and the state of the practice differ significantly, and the low industrial relevance of solutions proposed in the literature prevents a wider adoption of traceability for product lines in industry.

We present a study on the state of adoption of variability-management concepts and remaining practical challenges in twelve industrial cases from different organizations engineering variant-rich systems. We used document analysis, semi-structured interviews, and focus groups on cases that cover a wide range of domains and development scales, from a rather small web-application case to ultra-large software engineering for automotive or

industrial component production. In addition, we conducted a lightweight literature review on relevant SPLE adoption case studies, experience reports, surveys, and meta-studies, supporting the formulation and synthesis of challenges we present.

Our industrial cases primarily represent the development in a small part of a company, such as a single division or development team. We refer to these as *use cases* in the remainder. We also investigated cases provided by tool vendors who are looking to integrate product-line engineering concepts into their tools, referred to as *tool cases*. For our study, we combined eliciting structured case descriptions with focus-group interviews to identify the concepts that were adopted, as well as variability drivers and variable assets.

We report our cases' drivers of variability, the SPLE concepts they adopted, as well as remaining challenges to be addressed by the research community and tool vendors.

We believe our results support practitioners and researchers. Practitioners can use our results as a baseline to compare their own organization's capabilities and understand which concepts are at their disposal for future development. Researchers obtain the current state of adoption of SPLE concepts in industry, as well as they learn about remaining challenges.

2 Background

We briefly discuss strategies and important concepts for engineering variant-rich systems, gradually from ad hoc strategies to more advanced variant management strategies and concepts, partly inspired by the levels proposed by Antkiewicz et al. (2014).

Clone & own An organization creates variants by copying and adapting existing variants to new requirements. Assets are propagated in an ad hoc way among the variants. No platform or any kind of systematic variability management exists. Clone & own is a simple and readily available (Dubinsky et al. 2013; Duc et al. 2014; Businge et al. 2018; Stanciulescu et al. 2015) strategy for developing variants, but does not scale with the number of variants and easily causes maintenance overheads.

Clone Management Enhancing the governance, an organization could adopt a clone-management framework. Such frameworks (Rubin et al. 2012, 2013a, b; Pfofe et al. 2016; Antkiewicz et al. 2014) have been proposed, but have not found any documented adoption. They allow managing clones by using features as the main entities of reuse (instead of code assets) and record meta-data about the clones.

Configuration An organization can introduce configuration mechanisms to reduce redundancies. Such a mechanism is an implementation technique to realize calibration or variation. In the latter case, it is commonly referred to as a variability mechanism (Van Gurp et al. 2001; Berger et al. 2014c), ranging from simple conditional compilation (e.g., #ifdef) via control-flow conditional statements (e.g., IF), build systems (Berger et al. 2010a; Dietrich et al. 2012), and component frameworks to so-called feature modules (Apel and Kästner 2009) or delta modules (Schaefer et al. 2010), or combinations thereof (Behringer et al. 2017; Mukelabai et al. 2018a). The configuration options (a.k.a., calibration parameters or just parameters) and their constraints are typically declared in a model, such as a feature model (explained shortly). Using such a model, an interactive configurator tool can support the configuration process, guiding users by propagating choices or resolving configuration conflicts.

Platform Further scaling the development, an organization can adopt an integrated platform. There, instead of cloning, all variants are integrated into one software platform. By exploiting the commonalities among the variants, redundancies are removed while variability among the variants is represented by variation points within the platform. Variability is typically described in terms of features declared in a feature model, as follows.

Feature Using the notion of feature is core to scaling the development, for instance, when adopting a platform. A feature represents end-to-end functionality of a system (Berger et al. 2015). Features are intuitive entities that can be understood by different roles, including marketing experts, project leads, and customers. Features abstractly represent assets and are tracked, for instance, in a database or, more formally, in a feature model. Many features also serve as a configuration option (a.k.a., *optional* feature).

Feature Model Keeping an overview understanding of features, organizations should create a feature model—an intuitive, tree-like representation of features and their constraints (Kang et al. 1990; Czarnecki et al. 2012; Acher et al. 2013a; Nesic et al. 2019). The typical graphical notation is shown in Fig. 1, representing a small excerpt of the Linux kernel's variability model. For brevity, we refer to related work (Berger et al. 2010b, 2013b, 2014c) for an introduction into feature models and this particular example.

SPLE Process For an organization to effectively engineer a platform, textbook SPLE methods (Apel et al. 2013; Pohl et al. 2005; Czarnecki and Eisenecker 2000) introduce two main processes: domain engineering, which aims at engineering the platform, and application engineering, which aims at deriving individual variants from the platform. Each process comprises typical software-engineering tasks, such as requirements engineering, design, implementation and quality assurance, with domain engineering also containing scoping (determining and prioritizing platform features).

Product-Line Quality Assurance Enhancing the quality assurance, organizations can adopt analyses of product lines (Thüm et al. 2014; Benavides et al. 2010; Mukelabai et al. 2018b), which differ conceptually from single-system analyses. The latter, especially dynamic analyses such as testing, can only be used for individual variants. While they can be beneficial to optimize individual variants, single-system analyses are usually insufficient when a product-line platform has been adopted and errors related to all possible variants should be found. For instance, unwanted feature interactions can occur for certain variants based on the combination of features in the variant. Applying single-system analyses for finding such errors requires configuration sampling (Cohen et al. 2007; Perrouin et al. 2010).

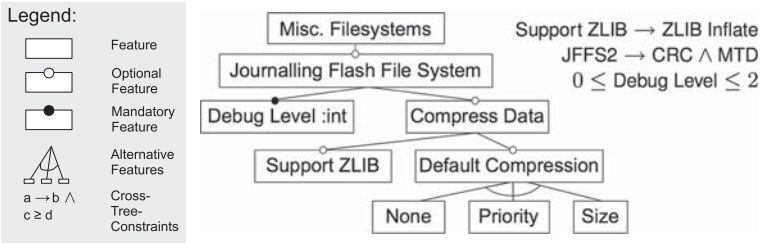

Fig. 1 Feature model example (Berger et al. 2014c)

3 Methodology

Our study aims to "collect and summarize evidence from a large representative sample of the overall population" (Molléri et al. 2016) and can therefore be constructed as a survey. We follow the well-known guidelines by Kitchenham and Pfleeger (2002) to structure this section, and we rely on the simplified 7-step process of Linåker et al. (2015).

3.1 Aims and objectives

As laid out in Section 1, the motivation for our study was to understand which variability management concepts are adopted in substantial industrial cases, and the challenges that industrial practitioners perceive in their daily work, determining the concepts that are still needed. Coming from an academic perspective, we also aimed to identify in how far the academic literature on SPLE takes industrial circumstances into account and to what extent the solutions offered by academia correspond to the practical challenges. Our discussions with practitioners, and our lightweight literature review, supported our view that there is a disconnect between SPLE practice and theory. We formulated three research questions:

RQ1: What are drivers for variability in our cases?
RQ2: Which SPLE concepts are adopted in our cases?
RQ3: What concepts are missing for our cases and for cases previously reported in the literature?

3.2 Planning, scheduling, and designing the study

Based on the research questions, we planned a descriptive study using qualitative data. The study was planned to be conducted in several iterations where data was collected repeatedly, then analyzed, and finally followed up. The data collection itself ran during 2018. In addition, we conducted a lightweight literature review on industrial case studies and experience reports on adopting SPLE, to triangulate with the challenges from our twelve cases.

Our personal network gave us access to companies that engineer variant-rich systems and intend to invest into improving their engineering. We therefore employed purposive sampling, in particular expert sampling (Etikan et al. 2016) in which we identified those experts that deal with the problems we set out to investigate. We created a list of potential candidates and narrowed it down by excluding those with insufficient resources to participate. We also selected companies that would provide a specific viewpoint and tried to achieve heterogeneity both in terms of company size, age, and size of the system, and implementation technology. We also achieved a good geographical distribution by including companies from five different countries in the European Union.

The final list consisted of twelve companies, nine of which create a variant-rich system and three of which are tool vendors selling software-engineering tools that want to integrate variability mechanisms into their existing tools. For assuring anonymity, we report the cases we studied and only provide high-level information about the different companies, and the meetings we had with them.

3.3 Data collection

The primary data collection method was a *case description* provided by the companies that describes the case for product line engineering. It relied on a template that included the

product and market context, the technology context, a description of existing processes and automation techniques, goals, and key performance indicators, the variability management practices currently in use, and the principal assets, their reuse and management. The template was piloted with one of the organizations and subsequently refined. The organizations involved in this study iteratively improved the case descriptions to homogenize them and to ensure that relevant information was included.

Based on the description, we conducted a preliminary analysis to identify open questions. This resulted in an interview guide for each of the cases that was used in *semi-structured interviews or in small focus groups*. While the interviews were conducted with experts from individual companies, the focus groups included representatives from several companies. We chose to use the latter format when cases were similar to each other and we needed to understand the differences and similarities better, and when resources allowed such a meeting. Each of these occasions included one or two researchers and at least two industrial participants. In all cases, the industrial participants were engineers working actively with software product lines. Each interview or focus group lasted between 30 and 60 minutes and was conducted by the researchers in person. Data was collected in the form of extensive notes and shared among all authors. The semi-structured format allowed us to explore additional aspects that were not covered in the original case descriptions. Specifically, we had the following number of small focus-group meetings or interviews: power electronics (3 meetings, 4 participants), traffic control (1 meeting, 1 participant), chip modeling (2 meetings, 3 participants), modeling platform (2 meetings, 2 participants), railway (8 meetings, 3 participants), aerospace (3 meetings, 4 participants), truck manufacturing (2 meetings, 3 participants), web application (1 meeting, 2 participants), automotive firmware (2 meetings, 3 participants), requirements engineering (1 meeting, 1 participant).

On two occasions, we held larger *focus group meetings* with a majority of the involved researchers and companies. This assured that we met representatives of all companies at least once. We discussed adopted concepts and needs, providing another opportunity to explore differences and similarities.

We also conducted a lightweight literature study where we collected and inspected meta-studies, surveys, exploratory studies, industrial case studies, and experience reports. Even though, we know from experience that the large majority of these publications focus on praising the benefits of SPLE (e.g., cost savings and shorter time to markets) while not providing sufficiently detailed data on the adopted concepts and on challenges (i.e., remaining challenges for SPLE research, as opposed to those that were solved in the case study), we decided to systematically collect those publications and triangulate our synthesized challenges with those from these cases. To collect publications, we used our own expertise as well as we consulted the SPLE community's "Hall of Fame,"[1] a book with a collection of successful SPLE cases (van der Linden et al. 2007), and the Software Engineering Institute's catalog of case studies (Software Engineering Institute 2008).

3.4 Data Analysis

We analyzed the data in several iterations. The first iteration based on the use case descriptions was mainly targeted at identifying which aspects of the current product line approach in the cases remained unclear—to be able to follow-up with our industrial contacts. For this purpose, we transformed each use case into a narrative as presented in Section 6. This

[1] http://www.splc.net/fame.html

allowed us to structure the information and find aspects that needed clarification. We then derived questions for semi-structured interviews and small focus groups from this. The results also guided the discussion in the larger focus groups. We met regularly to discuss the open questions and to gain an overall understanding of the issues.

We then merged information from the interviews and the small and large focus groups in the common narrative. In addition, we performed coding on the available data. The codes focused on the current practices and the challenges stated by the companies. We pre-defined codes based on variability management concepts and known challenges from the literature, and complemented these with emergent codes that were based on the first analysis round and the information from the interviews and focus groups. Once a stable set of codes emerged, we conducted a coding workshop to harmonize and refine the codes. Again, all information available at this point was used to validate the codes, to join codes with a high degree of similarity, and to refine the codes, in particular with respect to their concrete formulation. If questions arose during the data analysis or the researchers disagreed on the data, we used personal contacts within the organizations to clarify issues quickly before misunderstandings could arise or bias could manifest.

Once the concepts were identified and the authors agreed on the adopted concepts, we forwarded the information to our contacts for member checking. We also asked clarifying questions about the overall study and the long-term perspective of the cases.

For the literature review, we analyzed the collected publications by reading through the paper, specifically searching for challenges related to variability management concepts that were not solved within the respective case. We mapped those challenges to our challenges, enriching the challenge descriptions. Of course, the validity of reporting challenges from other cases, most of which are at least one or two decades old, is lower than the challenges we extracted from our companies. Still, they substantially enhance the relevance and richness of our reported challenges.

4 Literature review

In our lightweight literature review we identified the following related meta-studies, surveys, case studies, and experience reports.

4.1 Meta studies

Marimuthu and Chandrasekaran (2017) conduct a tertiary study of systematic studies on variability management. They provide detailed bibliometrics, but do not extract any challenges that might be reported in their identified studies. Bastos et al. (2017) investigate product-line adoption in small and medium-scale companies via a multi-method approach comprised of a mapping study, a case study, and a survey among experts. They mainly elicit success factors and practices, but no challenges. Chen et al. (2009) conduct a literature review on variability management and, among other results, list the challenges evolution ("systematic approach to provide a comprehensive support for variability evolution is not available"), scalability of techniques, as well as testing and quality assurance in general. In another study, Chen and Babar (2011) study the state of evaluation of variability management techniques, concluding the lack of proper evaluations as a challenge that researchers should address. Finally, Mohagheghi and Conradi (2007) conduct a literature study on the benefits of software reuse (not limited to reuse via variability management), emphasizing that: "For industry, evaluating reuse of COTS or OSS components, integrating reuse

activities in software processes, better data collection and evaluating return on investment are major challenges."

4.2 Exploratory studies

Chen and Ali Babar (2010) present an exploratory study relying on focus-group research investigating the perceived challenges of variability management using eleven participants from organizations that do consulting or in-house SPLE. With respect to variability modeling, the focus group reports, among others, the following challenges: visualization of features, evolution and maintenance of models (in particular dependency management), and variability modeling being not very user-friendly. In general, the study questions academic techniques. It also points out challenges when migrating to SPLE (cf. Section 7.3), specifically that clone detection techniques are not applicable to multiple systems. Furthermore, while structural variability is well supported, behavioral and timing aspects are not.

4.3 Surveys

A survey of variability modeling in industrial practice by Berger et al. (2013a) lists specific challenges for variability modeling, including visualization, model evolution, and traceability. Thörn (2010) survey variability management in small and medium-scale companies in Sweden; however, the reported challenges are rather general and not specific to variability-management concepts.

4.4 Experience reports and case studies

Over the last decades, practitioners and researchers have published a large number of case studies and experience reports, the majority between the end of the 1990s and the beginning of the 2000s. We now list all cases we identified. When available, we provide all references that describe the case in detail. However, some cases were described in the respective source only, not as part of a publication on its own.

All the cases in the book of van der Linden et al. (2007) describe successful adoptions of product lines, where usually the typical concepts are adopted (platform, feature modeling, automated product derivation, automated testing); for each, the obstacles and limitations in SPLE are also described. We inspected all the cases: AKVAsmart, Bosch (Tischer et al. 2011; Steger et al. 2004; Thiel et al. 2001), DNV Software, MarketMaker (Verlage and Kiesgen 2005), Nokia Mobile Phones, Nokia Networks, Philips Consumer Electronics Software for Televisions, Philips Medical Systems, Siemens Medical Solutions, and Telvent. Furthermore, the book also referenced the following cases, each of which we inspected as well: Salion (Buhrdorf et al. 2003; Clements and Northrop 2002), Testo (Schmid et al. 2005), Axis and Ericsson (Svahnberg and Bosch 1999), Axis and Securitas (Bosch 1999a, b), and RPG Games (Zhang and Jarzabek 2005)

From the SPLE community's "Hall of Fame"[2] we identified and inspected: Boeing (Sharp 1998), CelsiusTech Systems AB (Bass et al. 2003; Brownsword and Clements 1996), Cummins (Clements and Northrop 2001b), Ericsson Telecommunications Switches, Fiscan Security Inspection Systems (Li and Chang 2009), Hewlett Packard's printer

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

firmwar Owen (Toft et al. 2000), HomeAway (Krueger et al. 2008), Lockheed Martin, LSI Logic (Hetrick et al. 2006), Lucent, Siemens Healthcare (Bartholdt and Becker 2011), Toshiba (Matsumoto 2007), U.S. Army (Lanman et al. 2013), U.S. Naval Research Laboratory (Bass et al. 2003), General Motors (Flores et al. 2012), and Danfoss (Jepsen and Beuche 2009; Jepsen et al. 2007b; Fogdal et al. 2016). The cases Salion, Bosch, MarketMaker, Nokia, and Philips (Medical Systems and Software for Television Sets) were already contained in the book of van der van der Linden et al. (2007), explained above.

We inspected all cases of the Software Engineering Institute's catalog of case studies (Software Engineering Institute 2008): US Army's Common Avionics Architecture System (CAAS) (Clements and Bergey 2005), CCT (Control Channel Toolkit) (Clements et al. 2001a), Naval Underwater Warfare Center (Cohen et al. 2002, 2004a, b), Argon (Bergey et al. 2004), ABB (Ganz and Layes 1998; Rösel 1998; Pohl et al. 2005; Stoll et al. 2009), Deutsche Bank (Faust and Verhoef 2003b), Dialect Solutions (Staples and Hill 2004), E-COM (Liang et al. 2005), Ericsson (Mohagheghi and Conradi 2008; Andersson and Bosch 2005), Enea (Andersson and Bosch 2005), Eurocopter (Dordowsky and Hipp 2009; Hess and Dordowsky 2008), Hitachi (Takebe et al. 2009), LG (Pohl et al. 2005), Lufthansa (Chastek et al. 2011), MSI (Sellier et al. 2007), NASA (Ganesan et al. 2009), NASA JPL (Gannod et al. 2001), Nortel (Dikel et al. 1997), ORisk Consulting (Quilty and Cinneide 2011), Overwatch Textron Systems (Jensen 2007a), Ricoh (Kolb et al. 2005), Rockwell Collins (Faulk 2001), Rolls-Royce (Habli and Kelly 2007), TomTom (Slegers 2009), and Wikon (Pech et al. 2009). The cases CelsiusTech, Salion (Clements and Northrop 2002), Axis (Bosch 2000), Boeing, Cummins, Danfoss, and DNV Software were already contained in one of the other sources above.

Finally, we also included some cases that we know, from our experience, are neither contained in the book of van der Linden et al., the SPLE community's "Hall of Fame" nor the Software Engineering Institute's catalog. These were: six German SMEs (including MarketMaker from above) (John et al. 2001), a telecommunication system known as Terrestrial Trunked Radio (TETRA) (Pohjalainen 2011), Volvo Cars and Scania (Eklund and Gustavsson 2013; Gustavsson and Eklund 2010), Audi (Hardung et al. 2004), and Daimler (Dziobek et al. 2008; Bayer et al. 2006).

4.5 Summary

While we will report the identified challenges from the literature together with our challenges in Section 7, we learned that the majority of publications does not report challenges that pertain specifically to variability management or that have not been resolved in the course of the respective case. Most case studies report practices or lessons learned that contributed to the success, but not challenges. Especially, all are about successful adoption, and primarily report about the perceived benefits (some also quantified) that SPLE brought. Negative experiences, shortcomings of tooling, or actual challenges for the SPLE community are largely missing. For most of the case studies and experience reports, we conjecture that these are biased, since the case study authors primarily want to show success stories instead of problems and failed attempts. As such, most of these publications primarily report on the benefits that were achieved, as well as they report success factors experienced as deemed relevant for SPLE. When reporting about the specific product line, the predominant focus is on the product-line architecture, followed by organizational and process aspects. Interestingly, some publications even have "challenges" in the title, but those challenges are usually experiences and hindrances that the organization faced before adopting SPLE or that occurred during the case and that were addressed.

Furthermore, it is apparent that some challenges mentioned in previous case studies have been addressed nowadays. For instance, for Bosch (Tischer et al. 2011; Steger et al. 2004) and MarketMaker (Verlage and Kiesgen 2005), the publications emphasize the lack of proper, industry-strength SPLE tooling, including feature modeling, as the main challenge, which is addressed with commercial and open-source tools nowadays. Also, Chen and Ali Babar (2010) report that variability modeling is not very user-friendly, which can be seen as a solved challenge with the commercial and open-source feature-modeling tooling that exists nowadays.

Finally, we also observed that the extent and level of detail in which challenges relevant for SPLE researchers are reported is not sufficient. Some authors provide information about adopted concepts, for instance, as van der Linden et al. (2007) point out for their collection of cases: "Most architectures are based on a platform, supporting the requirements of present and future products. Often there are several similar products that are combined in the product line to improve the benefit of reuse. The development of a common, variable platform is often considered as the basis for introducing the product line in the organisation. Plug-in mechanisms and the definition of the right interfaces seem to be crucial." The majority of publications does not provide a finer level of details.

5 Variability drivers and variable artifacts in our cases

We begin the report on our results by discussing the factors driving the variability in our cases in Section 5.1. These variability drivers affect various types of artifacts, which we discuss in Section 5.2. This data has been derived from the data we collected in the case descriptions, interviews, and focus groups.

5.1 Primary variability drivers

For our cases, a number of different variability drivers was reported, as shown in Table 1. The most prominent drivers are *markets* and *hardware*. Being able to place products on different markets with different regulations and to ensure that the products are able to adopt to new market needs is crucial. Hardware is a relevant driver, since many of our cases concern systems, and the software needs to be able to work with a variety of different target hardware. In many cases, it is the customer who can select certain hardware, and the software needs to be able to run on the hardware configuration chosen by the customer. This is one form of *end-user customization*, another important variability driver.

> *In times of Industry 4.0 and IoT the customers demand connected and smart drives. So, the variability of features within future drive generations will be quite high.*
>
> — power electronics (use case)

An increase in variability through a number of forces was also reported. One case of firmware for power electronics, for instance, sees an increased need for variability driven by the more wide-spread use of different types of multi-core processors in their products (*hardware*) and increased industrial digitalization (*markets*, *operating environments*). The importance of the market and its growth as the prime driver of variability is also mentioned for the automotive firmware and traffic control cases. The latter also emphasizes that

Table 1 Variability drivers and variable artifacts as reported by the different cases

	use cases									tool cases		
	power electronics	truck manufacturing	aerospace	automotive firmware	railway	web application	modeling platform	imaging technology	traffic control	requirements engineering	hardware modeling	chip modeling
Variability Drivers												
hardware	●	●	●	●	●			●	●		●	●
usage scenarios	●			●				●				
markets	●	●	●	●	●	●	●	●				●
simulation			●[1]					●				
end-user customization				●	●	●	●	●	●			
operating systems and middleware					●		●		●			
operating environments	●						●	●	●	●		
evolving requirements						●						●[2]
design exploration											●	
Variable Artifact Types												
source code	●	●	●	●	●	●		●	●	●		
build system	●											
custom descriptors	●[3]		●[4]	●[3]			●[7]	●		●[8]	●[10]	●[11]
system models	●	●	●		●			●	●	●		
interfaces		●						●	●			
tests						●	●	●		●		
meta-data							●[5]			●[9]		
requirements					●		●	●	●	●		
components							●[6]	●	●			
operating system and base libraries								●	●			

[1] the need for simulations substantially increases variability, since models represent environments or hardware at different levels of fidelity

[2] functional requirements, such as processing power or connectivity, and non-functional requirements, such as power consumption and safety as well as intellectual property (IP) cores

[3] XML files as source for code generator

[4] proprietary DSML instances

[5] documentation, language files, resources

[6] OSGi bundles

[7] meta-models and UML profiles

[8] software, hardware, and simulation models

[9] documentation

[10] design variants as proprietary DSML instances (graphs) in XML files

[11] proprietary DSML instances based on the IP-XACT standard (IEEE1685), with some in-house extensions for modeling registers and memory

innovation is an important driver for variability: the company needs to be able to deliver innovative solutions while at the same time be able to maintain the existing products in the portfolio. For our modeling platform case, the organization explained that the product needs to compete with software-as-a-service (SaaS) offers, where customization is seen as an advantage. Furthermore, *IoT* is a new, core driver of variability, as prominently mentioned for the cases power electronics (more precisely, Industrial IoT) and chip modeling.

Another interesting driver is *simulation*. In our aerospace case, a simulator resembles the real aircraft, but has more variability through the use of models at different levels of fidelity. For instance, verifying a specific sub-system might require a detailed high-fidelity model, while for real-time simulation, the model needs to be replaced with a lower-fidelity model (which might use interpolation) due to limits in computation capability.

5.2 Variable artifact types

Not surprisingly, the most frequently mentioned variable artifact is *source code* as shown in Table 1. Many companies use conditional compilation with preprocessor directives to include variability information in the source code. *Custom descriptors* are also relatively common, for instance, as the foundation for code generation. These are often expressed using domain-specific languages. We found little evidence for variability in *tests* and *requirements*. Only one company explicitly reports to use *components* as variable assets, but we expect that there are many companies that do this implicitly.

A use case that is a bit neglected in research is variability in models used for code generation. Five of our subjects write application logic in Simulink and then generate code. Apparently, common variability-management techniques on the code level are not applicable; instead, variability modeling concepts, especially variation-point support is needed in the models and needs to be supported by the modeling tools.

6 Adoption of variability management concepts in our cases

We now introduce our use and tool cases by describing core characteristics and the adopted variability management concepts. An overview can be found in Table 2, which also shows the near-term adoption goals. The particular challenges faced by the organizations will be presented thereafter, in Section 7. Each case description follows a common format: context, variability drivers, variability strategy, additional capabilities (e.g., traceability or testing), and product derivation.

6.1 Power electronics use case

This case concerns the production of, among others, motor controllers (drives) for electric motors. Around 300 million drives are in industrial use worldwide and used in mining, ski lifts, big industry automation processes, and turbines (e.g., solar and wind turbines). The development is characterized as agile through clone & own.

The diversity in hardware and usage scenarios is the primary driver of variability. In addition, country-specific regulations contribute to the number of required variants. The company expects a further increase in variability through trends such as multi-core processing and increased industrial digitalization, as well as adding more software features to the drives.

> *So far [the variants] are managed in a clone & own manner, but that will not be possible in the future [...] going to multi-core, digitalization or adding more features [...].*
>
> — power electronics (use case)

This use case primarily relies on *clone & own*. Yet, *configuration mechanisms* in individual variants also exist. The drive software is written in C/C++ and a substantial part of the code is generated from XML files with an in-house generator. The source code contains *preprocessor statements* controlled by *configuration options*. Automatic testing of individual variants is in place through a continuous integration system using Jenkins with automated, nightly tests. The *connection between customer adaptations and features* is tracked in a database to ensure long-term maintainability. Likewise, *rationales* for variability decisions are recorded.

Table 2 Adopted product-line engineering concepts and near-term adoption goals

	use cases									tool cases		
	power electronics	truck manufacturing	aerospace	automotive firmware	railway	web application	modeling platform	imaging technology	traffic control	requirements engineering	hardware modeling	chip modeling
clone & own	●		●[1]	●	●	●		●	●	●[11]		●[11]
clone management			○[1]					○	○			
configuration												
component/module selection	●	●	●	●	●	●	●	●	●		●	
conf. options / calib. parameters[15]	●	●	●	●	●	●	●	●	●		●	
configurator tool		●		●			●	●	○			○
features[2]	○	●	●	●	○	○	●[13]	○	○	○	○	○
centralized variability representation[14]	○	○	○	○	○	○	○	○	○	○	○	○
feature model												
FODA-like model	○	○	○	○	○	○				○	○	○
feature database		●		●			○					
informal, but structured feature model[4]			●	●		●	◒	●	●			
declared/managed feature constraints	○	●[6]	○	○	○	○	●[6]	●	◒	○	○	○
platform												
integrated software platform	○	●	●	●	○	○	●	●	◒	○	◒	○[10]
product (variant) derivation	○	●	○	●	○	○	○	◒	◒	○	○	○
integrated with SPLE tool			○	◒	○	○	○			○	○	○
separated domain and application engineering		●		●	○		●	◒	○	○	○	
variability-aware analyses												
feature-model analyses	○	○	○	○	○		○	◒	◒	○		○
visualizations (e.g., feature hierarchy)	○		○	○			○	○	○	○	○	○
change-impact analyses[3]								○	○			
consistency analyses	○	●	○	●	○		○		○	○		○[16]
requirements completeness/correctness		○								○		
safety analyses		○										
performance analyses				◒				●	●			
other quality attributes analyses							◒	●	●			○[18]
traceability												
feature-to-code	○	●	○	●	○	◒	◒	●	◒	○	○	○
customer adaptations	●		○	●	◒	●	●	◒	◒			
variability decisions/rationales	●			●	○		○	○	○			
security												
feature-level authorization			○				◒	◒	◒	○		○

●adopted

◒rudimentarily adopted

○adoption goal

[1]on product-line level

[2]or feature-like entities

[3]on variant level

[3]tool vendor, applies to usage of tool

[4]e.g., spreadsheet

[6]declared over assets

[10]a platform of models

[11]tool vendor, applies to usage of tool

[13]Eclipse features; very high level

[14]also unified to some extent

[15]for in-component/in-module configuration

[16]constraints checking

[18]Systems-On-Chip specific quality attributes such as heat dissipation or power consumption

The specific variants are built through a Python-based build system that allows *component selection*.

6.2 Truck manufacturing use case

This case comprises the software development of a large truck manufacturer. All of its products (80,000 trucks per year) come from the same platform.

Almost every product shipped has a unique configuration. The main differentiator of the brand on the global market is full product customizability.

The truck manufacturing case relies on an *integrated platform* to manage variability and employs *separate domain and application engineering*. All *configuration options* are realised with configurable values (i.e., without #ifdef or similar constructs).

Features are maintained in a *feature database* that contains *traceability links to the code*. Assets (different levels of specifications and source code) are stored in separate databases. Consistency is checked when a system is made ready for release.

> *Even a simple fuel level indication system in a truck has 24,000 variants! This means that only a very small portion of all possible variants can possibly be verified by testing.*
>
> truck manufacturing (use case)

On release, all relevant information about a system, including trace links and variability information (presence conditions) are released to the product data management (PDM) system. When deriving a product from the PDM, a specialized *configurator selects the relevant components* and derives source code parameters to generate the source code variant that implements the desired functionality using the possible values for configuration parameters and the constraints as input.

6.3 Aerospace use case

This case is related to the development of an aircraft simulator for a full, configurable aircraft. Both the simulator and the aircraft software can be seen as product lines.

Variability is driven by differences in equipment and software between aircrafts. Assets are primarily developed for the aircraft product line and then propagated to the simulator product line via *clone & own* of the entire product line. For the simulator, multiple variants target different scenarios ranging from simple computer-screen-based simulators to realistic simulator with actual cockpit hardware. We focus on the simulator.

> *Before the migration [to a product-line approach] it was estimated then by the product managers that we would save 6.8 million euros in three or four years . We have saved more than that.*
>
> aerospace (use case)

The simulator is an *integrated platform* with *features* that can be mandatory, optional, or a special type of optional features that need to be deletable without a trace to protect customer interests. Each of the latter is completely modularized, meaning that the complete module can be left out and that none of these features is cross-cutting. Another type of feature, so called "role change equipment," are customer-configurable features that are placeholders for future development. This means that configurations can be partial and are selected with *component/module selection* at checkout time. The organization maintains an *informal*

feature model as a spreadsheet with a hierarchy, but no explicitly modeled dependencies, as well as manifest files describing components. In addition, *calibration parameters* refine components.

To derive a product, these parameters are set at build time. Preprocessor directives are prohibited.

6.4 Automotive firmware use case

This case concerns a complex product line for electronic control units (ECUs). Around 2,000 variants are delivered per year; each deliverable is a distinct configuration. Each variant can comprise over 100 function packages and up to 2,000 functional components; the latter are updated regularly (every three months).

Variability arises from the need to customize ECUs to different vehicle types and customers. An *integrated platform* was defined to reduce time-to-market of variants for customers. It is combined with a limited version of *clone & own*, since developers can choose to create a new branch for a new feature based on guidelines that take longevity and complexity of the feature into account. Packages are developed as part of the platform (*domain engineering*), but variant- and customer-specific packages can exist (*application engineering*). The functional components can be configured via static parameters, which are *feature-like entities* used within variation points relying on *conditional compilation* (e.g., with #ifdefs).

These parameters are arranged in a relatively flat hierarchy stored in a distributed *feature database* and allow *feature-to-code traceability*. *Rationales* are recorded by tracing feature information to requirements. An *informal feature model* maps the high-level features and the static parameters. Constraints are defined over the static parameters. Interestingly, these parameters represent both variations and versions, since some of them map to pre-processor macros, and the *version-control system directly supports checking out variants*. As such, variability and version management are to some extent unified (like in variation control systems Linsbauer et al. 2017; Stanciulescu et al. 2016; Berger et al. 2019a).

> *We are [...] outstanding in managing product variants [...] Of course there is further potential for improvement and that is what we are seeking.*
>
> — automotive firmware (use case)

Also, dynamic (calibration) parameters for late binding exist, some of which can be defined by the customers after delivery. These parameters, including preconditions, are stored in separate databases upon which experts configure *customer adaptations* using in-house tools. Configurations are recorded in a special database.

Various analyses are run on the distributed feature database, including internal feature consistency checking. An SPLE tool is increasingly used to complement the existing cross-database consistency analyzes with rules and predicates. Single-system performance analysis is done for the products shipped to customers.

6.5 Railway use case

This case concerns the development of signalling systems for urban transport networks for many cities in the world. Each city has specific needs for the signalling system, which is reflected in the variants.

The specific signalling system for each city is created from a different city's variant using clone & own by creating branches in the software repository. The variants are composed

by *component selection*, where components represent modules for different sensors and functionality. In addition, each variant also contains a set of internal *configuration options* based on C preprocessor directives.

> *Developers tend to be specialized in one variant [...], as the architecture is different [...]. They have difficulties to switch from one variant to another. Due to these difficulties, merges are done by the most experienced developers, who we would want to use on more useful tasks.*
>
> railway (use case)

The software was refactored and broadly re-architected at least in one branch. Due to development constraints (e.g., time, budget, separate responsibilities), the company never had time to integrate these branches. *Customer adaptations* are tracked, since each client is represented by a specific branch.

6.6 Web application use case

This case focuses on the creation of web applications that, among others, allow companies to manage media campaigns. Each client receives a customized variant.

Different variants are created with clone & own. Each application consists of frontend and backend code and most variability is in the user-facing frontend part of the system, realized using JavaScript and AngularJS. Features are recorded in a highly *informal feature model*. Otherwise, the company relies on the knowledge of an expert engineer who knows the distribution of features across branches. The system is implemented using different services where a final product is a composition of different services from a core library, product-specific code, and third-party services. The core services can be adapted with *calibration parameters* in configuration files to achieve specific behavior for each product. *Component selection* is performed on the service level.

This architecture allows *customer adaptations* through the services and provides limited *traceability between features and code*, since features are mapped to services.

> *The most common adaptation in variants are UI customizations. Since the web UI is not standard, users usually want a special interface.*
>
> web application (use case)

6.7 Modeling platform use case

This case concerns a commercial model-driven engineering tool for business architects, system architects, and developers. The tool is component-based and either comes pre-packaged for one of these target audiences or can be assembled based on customer wishes. It is also possible that specific features are developed for a certain customer. Additionally, different license schemes can be employed (e.g., commercial and open source versions), and six different operating systems versions are supported, further driving variability. There are twelve solutions, with two major releases per year, that include more than 50 modules and three meta-models. More than 30 modules with variations can be obtained from a store to fit specific needs. The store also contains scripts and model components for different versions of the platform.

To this end, an *integrated software platform* based on Eclipse plug-ins is used. All provided solutions share a common set of modules. Because features are localized in specific plug-ins, *feature-to-code* traceability is available. Currently, the variability in terms of packaging is not managed in a tool-supported way.

Variants are composed by *component selection*, relying on the Eclipse P2 packaging system, which considers constraints between features and plugins, and uses a *limited feature model*. The *configurator* defines which modules are part of a package and has limited support to set *configuration options* for the individual plug-ins.

> *No particular tool is used to manage the variability in terms of packaging. Technically the variability is ensured by the modular design.*
>
> — modeling platform (use case)

6.8 Imaging technology use case

This case comprises camera software that is packaged for individual customers based on customer requirements. This packaging includes configuring sensor parameters, prototyping and testing different sensor configurations, integrating onto the hardware for sensor validation and exporting sensor configuration parameters for use in a production software system.

An *integrated platform* along with a home-grown *configurator* tool is used to create these packages. Assets are currently managed via repositories (e.g., Git) and file-based storage. To create a product, *module selection* is used to package the correct firmware and driver software. In addition, *calibration parameters* are used to define the correct sensor and software configuration.

> *The key trend [we want] to follow [...] is mainly applying "continuous integration" to software infrastructure that supports the development and validation of hardware.*
>
> — imaging technology (use case)

6.9 Traffic control use case

This case concerns the development of several generations of integrated road traffic control and surveillance solutions, including custom sensors along with embedded software. Backend software is produced to process the data sent by these sensors. Different variants of the product are created for different customers and different regional markets, in particular if certification is necessary.

> *The challenges of the market are mostly technological. A relatively small group of players constantly improves the performance and versatility of the product. Regional presence is important in the market.*
>
> — traffic control (use case)

Ad hoc reuse via *clone & own* is present, relying on branching in the version-control system, which the company considers bad practice and strives for an integrated platform. If a software needs certification it can become a permanent branch in the software repository. *Calibration parameters* are used to configure the software for specific sensors. Some of

these are considered features exposed and sold to customers. *Component selection* happens at compile time where source code modules are statically linked or at runtime via dynamic linking.

6.10 Requirements engineering tool case

This case is our first tool case, concerning a tool for improving the quality of requirements. It allows defining an ontology of the application domain. Based on the ontology, clients can write semi-natural language requirements. The company also develops pattern-based extractors to populate the ontology from semi-structured documents (conceptually similar to techniques that offer languages for defining patterns that can be used to extract requirements from semi-structured documents Rauf et al. 2011). The clients of requirements engineering produce safety-critical software-intensive systems mostly in the transportation and defense industries.

> *One challenge faced by [our tool customers] is to take advantage of all the knowledge developed among different projects, produced in many times with no management in a disorganized way, and to reuse the existing requirements in new variations of the same systems.*
>
> requirements engineering (tool case)

Currently, the tool customers apply *clone & own* of the requirements specification. The tool does not offer variability features out-of-the-box. However, one stated use-case is the extraction of variability information — including a vocabulary and ontology of variability and assets — from requirements about an existing product-line platform. In other words, the requirements describe variation points and variants using dedicated terminology to be extracted by pattern-based extractors.

6.11 Hardware modeling tool case

This case concerns the development of a tool that allows to assemble system models (e.g., about automotive suspensions) from pre-defined building blocks and simulate them. The tool interfaces with other tools such as the company's own product-lifecycle management tool. The variability in the considered models concerns the blocks that vary, e.g., among automotive suspensions.

The current tool uses an ad hoc representation of variability only visible in terms of *component selection* inside the architecture models. The modeling language allows realizing variability encoding (von Rhein et al. 2016), that is, using built-in conditionals relying on configuration options, which enables this component selection. The tool is already able to explore the design space by creating all possible architecture models that are supported by the selected components. Nevertheless, the current tool chain does not support product line concepts explicitly. In particular, the concept of feature is not supported and variability is only managed within the design space without any knowledge about the problem space.

> *One challenge faced by [our tool customers] is to design and compare a significant number of product [...] design variants.*
>
> hardware modeling (tool case)

6.12 Chip modeling tool case

This case concerns a tool for designing Systems-on-Chips (SoC) and the reusable hardware component designs from which these SoCs are assembled. These designs are passed to machines that produce the corresponding integrated circuits. The company believes that emerging applications with huge growth potential, such as SoC for IoT will lead to a combinatorial explosion of variants, with features being related via complex trade-off constraints.

The variants differ by: their provided functions, execution performance of functions, packaging of SoC inside a circuit, power consumption, safety level (typically determined by standards), and life-cycle durations.

The modeling tool is based on the IP-XACT standard (IEEE 1685) and extensions for modeling registers and memory. The *integrated software platform* currently does not support any specific variability management facilities, nor is it seamlessly *integrated with an SPLE tool*. Consequently, the tool users are currently restricted to *clone & own*, specifically, using branching and merging facilities of the tool's built-in version-control facilities. Also, there is no *centralized variability representation*.

> *[There are] more and more applications with specifics—for example, the emerging IoT applications that require smaller platforms, with low or medium processing, and very hard constraints on reliability and power consumption.*
>
> — chip modeling (tool case)

7 Variability management challenges

We now synthesize and discuss the challenges faced among our cases. We observed that our cases cover a wide range of maturity levels with respect to the adoption of variability management concepts. Consequently, we structure the challenges according to these maturity levels, providing the context in which the challenge occurs and in which it should be addressed by researchers or tool builders.

We first present two general challenges that affect any maturity level. We then discuss those related to *support for clone management*, which are encountered in organizations that use clone & own as their main technique to derive new products. We then identify challenges that occur when *migrating to an integrated platform*, that is, when clone & own starts to be complemented by feature and asset management. Next, we discuss challenges that exist once migration is more or less complete and *working with an integrated platform* becomes the focus of work, followed by challenges that appear when the product line is *modernized and evolved*.

7.1 General challenges

***Challenge 1*, Model-Driven Engineering** A common challenge we observed for any maturity level is *model-driven engineering (MDE)* and *code generation*. While source code is still the most frequently mentioned variable artifact (cf. Section 5.2), we observed that often code is generated from domain-specific languages (DSLs). While the relation between DSLs and SPLE has been studied (Völter and Visser 2011), the DSLs used in our cases do not support SPLE concepts. As seen in our power electronics, aerospace, and truck manufacturing

cases, embedded systems organizations often rely on MDE using Simulink with code generation. Such a setup challenges not only handling cloned variants (e.g., when trying to trace features), but also adopting, working with, and evolving an integrated platform (Kolassa et al. 2015). This observation is substantiated by our three tool cases striving to integrate variability mechanisms into the modeling tools.

Furthermore, the experience reports on Danfoss (Fogdal et al. 2016), General Motors (Flores et al. 2012), and CCT (Clements et al. 2001a) also mention this challenge. They request a better integration of SPLE concepts, specifically variability mechanisms, with modeling tools. Notably, the experience report on Daimler (Dziobek et al. 2008) focuses on handling variability in Simulink models, specifically, it describes how to represent variation points in models. Nokia Networks (van der Linden et al. 2007) laments missing support for reusing systems engineering assets, which is in line with the experienced needs of customers of our our two tool cases chip modeling (Section 6.12) and hardware modeling (Section 6.11).

***Challenge 2,* Tool Integration** These elaborations from our cases and the literature illustrate a more general problem: *tool integration*. Since variability is a cross-cutting concern, variability-related tooling usually needs to be integrated with other engineering tools as named, for instance by our automotive firmware and aerospace cases that exhibit overall high maturity. This tool integration challenge was also expressed in the literature for Danfoss (Fogdal et al. 2016), General Motors (Flores et al. 2012), Siemens Medical Solutions (van der Linden et al. 2007), Fiscan (Li and Chang 2009), Argon (Bergey et al. 2004), and CCT (Clements et al. 2001a), who not only request integrated tool chains, but more mature variability-related tooling in general. General Motors (Flores et al. 2012) suggests to integrate SPLE concepts with product lifecycle management (PLM) concepts. Argon (Bergey et al. 2004) requests the integration of variability in version-control systems. In summary, this challenge is further supported by a recent study on product-line analyzes (Mukelabai et al. 2018b), which found that adopting such is, among others, a tool-integration problem.

7.2 Clone management

As seen in Table 2, seven of our use cases exercise clone & own for managing variants—while for two of our tool cases, the customers' variant management also relies on clone & own. The other tool vendor's modeling tool (cf. Section 6.11) offers variability encoding (von Rhein et al. 2016) using built-in conditionals relying on configuration options. While clone & own is the most common strategy for engineering variants, we observed substantial awareness among our subjects of the problems connected to it.

***Challenge 3,* Visualize and Track Variability** A need expressed by all cases is keeping an overview understanding of cloned variants, since understanding the content and purpose of individual variants is challenging without more abstract representations of the codebases.

Feature-Orientation Adding the notion of features and feature locations to clone & own would enhance the practice. When features are present, then stakeholders can know what is in the branches, as opposed to trying to understand the difference between variants through assets differences, which is challenging. In addition to recording features, recording their location, for instance, through lightweight code annotations (Ji et al. 2015; Andam et al. 2017; Abukwaik et al. 2018; Entekhabi et al. 2019; Krueger et al. 2019b) also helps with

maintenance. As such, this challenge is about keeping clone & own, but giving developers more control and overview understanding, which will help with maintenance (e.g., developers know the, potentially scattered (Passos et al. 2015, 2018), locations of a feature when cloning it to another variant).

In addition, better visualization and support for code propagation, ideally based on features, is needed. Especially for the power electronics case, better support for localization and propagation of features across cloned variants is requested. In summary, introducing some notion of feature-orientation to facilitate a more abstract understanding (abstracting over code-level structures) of variants, is an expressed need for our cases.

The existing case studies on Axis and Securitas (Bosch 1999a, b) emphasize the lack of architectural abstractions (features) in programming languages, as well as the exploratory study of Chen and Ali Babar (2010) confirming this challenge.

Record and Analyze Variability Decisions Several of our cases stated that they need a way to record and later analyze variability decisions (railway, imaging technology, and traffic control case), such as rationales for introducing variability. It was also stated that recording information about refactorings would be helpful, preferably using embedded annotations (Ji et al. 2015; Andam et al. 2017; Abukwaik et al. 2018; Seiler and Paech 2017; Krueger et al. 2018a, 2019a). The recording should explicitly relate refactorings to either the introduction of new functionality (features) or to maintenance. This information otherwise needs to be recovered when later propagating code across variants or migrating towards a platform (e.g., when assessing architectural mismatches). Specifically, for the railway and traffic control case, there is a need to reconstruct the architectural and functional evolution of the product line. While architectural evolution is mainly represented by typical refactorings (for the railway case, eight typical refactorings from Fowler (1999) are mentioned), functional evolution (i.e., the implementation of new functionality, bug fixes, and so on), needs to be distinguished.

***Challenge 4,* Cloning in Combination with Variability** An interesting observation is that none of our subjects exercises pure clone & own, but that variants already use variation points. This challenge is also mentioned in existing experience reports about Danfoss (Fogdal et al. 2016), AKVAsmart (van der Linden et al. 2007), Philips Consumer Electronics Software for Televisions (van der Linden et al. 2007), Philips Medical Systems (van der Linden et al. 2007), and Dialect Solutions (Staples and Hill 2004), as well as the case studies on Axis and Ericsson (Svahnberg and Bosch 1999). Interestingly, Staples and Hill (2004) point out for Dialect Solutions, and Fogdal et al. (2016) for Danfoss, that using variability mechanisms also avoids merge conflicts during clone & own and allows more isolated feature development.

Limitations of Clone-Management Techniques This challenge complicates using clone-management frameworks proposed in the literature (Pfofe et al. 2016; Rubin et al. 2012, 2013a, b; Antkiewicz et al. 2014) and techniques for integrating variants (Fischer et al. 2014; Martinez et al. 2015), since existing variability needs to be taken into account.

In fact, as explained for the railway case (cf. Section 6.5), the integration is done by experts who should rather realize new functionality instead of recovering information about variability in cloned variants and re-engineering code. Furthermore, most of our use cases applying clone & own also have an integrated platform, that is, a project with common assets. We are only aware of one work in this direction (Lillack et al. 2019), which should further be complemented with methodologies and tools.

Clone Management of Whole Product Lines Approaches to enable product lines of product lines have been investigated in the literature (Kästner et al. 2012; Rosenmüller and Siegmund 2010; Krueger 2006). However, what our cases require is *clone management at the product-line level*. A previous experience report on Philips Medical Systems (van der Linden et al. 2007) also emphasizes the reuse of components across product lines.

Our aerospace case exercises clone & own for two highly complex product lines. The primary product line controls a real aircraft, and the cloned product line the simulator. The case strives to improve the—currently manual and laborious—synchronization between aircraft and the simulator product line. Specifically, it requests guidance for creating variability models (e.g., whether one or separate, but largely redundant models should be created; how they should be decomposed to reflect the architecture) and modeling constraints. This should help defining an aircraft or a simulator configuration, including matching an aircraft configuration to a suitable simulator configuration for a particular test activity.

7.3 Migration to an integrated platform

Our three use cases that want to establish an integrated platform, as well as the customers of our two tool cases that aim to establish a platform, practice clone & own. Recall the core motivation for our railway case to free experienced developers from performing migrations (cf. Section 6.5). As such, migrations should be semi-automated, supporting less experienced developers or enabling domain experts performing it. We observed the following challenges.

***Challenge 5,* Platform Migration Process and Tools** For two of our use cases (power electronics and railway), the need for a dedicated migration process was expressed. Such a process should be lightweight and should guide engineers through the whole migration. Almost all of our use cases expressed the need for commonality and variability analysis. Furthermore, such a process should guide through the identification and location of features (potentially integrated with manual feature-location techniques Krüger et al. 2018b) or through creating the target architecture (potentially supported by feature-model composition techniques (Acher et al. 2010)). According to our railway and aerospace cases, such a process should also support engineers in creating variation points with an appropriate variability mechanism (cf. Section 2), in other words, prescribing the introduction of variation points. This challenge is also mentioned by previous experience reports on Bosch (Tischer et al. 2011), MSI (Sellier et al. 2007), and Siemens Healthcare (Bartholdt and Becker 2011), both requesting a process for incremental migration, the latter even during running projects, without disrupting the development.

Diffing of Cloned Variants We observed the need for higher-level diffing support for cloned variants. Specifically, as pointed out for our railway case, there should be means to express historical additions, suppressions, and modifications at the highest level of abstraction, specifically distinguishing architectural and functional evolutions. As such, practically usable, dedicated diffing techniques, which support existing variation points (e.g., #ifdef), are needed, potentially building upon recent techniques such as intention-based clone integration (Lillack et al. 2019). Enhanced visualization capabilities should also support highly scattered features (Passos et al. 2015, 2018), as explained for the web application and power electronics case. Furthermore, the visualization should be guided by extracting

the structure of the underlying configuration management base (including branching and revision structures), as pointed out for our railway case.

Notably, this challenge is also expressed for Ricoh (Kolb et al. 2005), and the exploratory study of Chen and Ali Babar (2010) reports that clone detection techniques are not applicable to software variants.

From a tool-vendor perspective, expressed for the chip modeling case, the organization strives to provide automated commonality and variability analyses to customers of the tool, so that existing cloned models can be migrated. As such, the challenge is to adopt automated analyses conceived by researchers, ideally adhering to standards, which are almost non-existent for variability management.

Asset Integration at Code and Model Level Integrating assets into a platform is challenging. Such an integration differs from integrating assets during clone & own in the sense that platform integration needs to consider many more variants—those derived from the platform—which might make it more challenging. Understanding and characterizing both kinds of integration is subject to future work. For our cases, better merge-refactoring techniques taking existing variability (e.g., #ifdef in the cloned variants) into account are necessary. For instance, our railway case needs such techniques for automatically proposing integration strategies for features from variants, that is, focusing on identifying functional (feature-based) evolution from the branching history, ignoring refactorings and other non-functional evolutions. Finally, asset integration is challenged (Challenge 1) when MDE techniques or code generation are used (e.g., in the power electronics case). This requires focusing on the migration (i.e., integration) of models, while not ignoring customized code.

Training, Certification, and Budgeting Other issues to be addressed by a process are to: (i) establish a common understanding among the stakeholders about product-line engineering concepts (requirements engineering case), calling for training support in a migration process, to (ii) establish certification support, as requested by an experience report on Rolls-Royce (Habli and Kelly 2007), and (iii) as expressed for our relatively small web application case, there is no dedicated budget to develop the platform, so assisting in budgeting is a challenge a platform migration process should support.

Definition of a Target Architecture A core issue when adopting an integrated platform is architectural degradation, and therefore architectural mismatches among the cloned variants, which need to be resolved. Architectures of the cloned variants need to be compared and a target architecture (Sinkala et al. 2018; Acher et al. 2011b) defined, which is expressed as a core challenge (e.g., for our railway case).

Specifically, expressed for the imaging technology case, the target architecture should be layered and modularized, to be maintainable. The goal is to enable fully automated variant derivation without programming effort (i.e., developing adaptations). Another challenge expressed is to create documentation (or generate such) about the architecture itself (chip modeling case), and about using such an architecture for variant derivation in parallel.

In our literature survey we found that most of the case studies focus on describing the architecture of the resulting product line. For instance, for RPG Games (Zhang and Jarzabek 2005), the authors discuss the architecture development and necessary adaptations of cloned variants (e.g., changes of variable types and refactorings) to obtain a common product-line architecture. Recently, Debbiche et al. (2019) and Akesson et al. (2019) provide datasets and experiences migrating cloned Java and Android game variants to such a common architecture.

***Challenge 6,* Migration Decision Support** A migration process should also provide decision support about the migration itself, based on the expected costs and benefits (Ali et al. 2009; Krüger et al. 2016). Specifically, for our power electronics case, the suggestion was made to enhance feature identification and location with indicators about the cost of the extraction and re-engineering of relevant assets for making them reusable or integrating into a platform.

Cost/Benefit Estimation The need for effective cost/ benefit estimation is further supported by existing experience reports on Deutsche Bank (Faust and Verhoef 2003b), on Philips Medical Systems (van der Linden et al. 2007), on Hitachi (Takebe et al. 2009), on Ericsson (Mohagheghi and Conradi 2008; Andersson and Bosch 2005), on Axis and Ericsson (Svahnberg and Bosch 1999), as well as on Axis and Securitas (Bosch 1999a, b). Specifically, for Philips Medical Systems (van der Linden et al. 2007), clone & own sometimes appeared to be beneficial (close to the break-even point), which led to tensions to develop outside the platform. Axis and Ericsson (Svahnberg and Bosch 1999) also need cost estimation for decision making. They explain it for the case that when a new product is added, sometimes rewriting the components instead of adapting them is easier, depending on the extent of the changes. The challenge is also described in other case studies on Axis and Securitas (Bosch 1999a, b), among others, demanding decision making support for: "Deciding to include or exclude a product in the product-line [...] Guidelines or methods for making more objective decisions would be valuable to technical managers."

Measurement The pre-requisite for cost/benefit estimation are effective techniques to measure the costs and benefits of reuse. Previous experience reports on Bosch (Thiel et al. 2001), Testo (Schmid et al. 2005), Argon (Bergey et al. 2004), Overwatch Textron Systems (Jensen 2007a), Wikon (Pech et al. 2009), and CCT (Clements et al. 2001a) emphasize such a measurement technique as an important challenge. Generally, in a case study on Axis (Bosch 1999a), the lack of economic models is lamented.

***Challenge 7,* Continuous Integration** Six of our cases (web application, traffic control, aerospace, automotive firmware, railway) explicitly point out the need for supporting continuous integration. Most of them have sets of automated unit tests (grown incrementally, typically with the discovery of bugs), UI tests, and integration tests. We found typical tools used for continuous integration, such as Jenkins, Maven, BuildBot, Git, and Jira. While some approaches to automate, e.g., interaction testing in continuous integration exists (Johansen et al. 2012), the larger challenge is to obtain a feature-oriented and configurable architecture that supports continuous integration, ideally with the tools mentioned.

This challenge is implicitly expressed in the cases for Philips Consumer Electronics Software for Televisions (van der Linden et al. 2007), for Overwatch Textron Systems (Jensen 2007a), and for CelsiusTech (Bass et al. 2003; Brownsword and Clements 1996), where balancing between domain and application engineering is a core challenge, striving to bringing both closer together. In addition, CCT (Clements et al. 2001a) demands a process for integration testing.

7.4 Working with an integrated platform

We observed the following challenges expressed for our use cases and tool cases when dealing with an integrated platform.

***Challenge 8,* Representation of Variability** A need expressed by all cases is a centralized (and ideally unified Berger et al. 2019a) representation of variability. For instance, for the aerospace and automotive firmware cases, an overall view providing a unified description of the variability was demanded, ideally in the form of a feature model declared in an established SPLE tool. Specifically, in one of these two cases, the variability information is scattered across different representations, including a distributed feature database and configuration files, preventing a coherent view of all relevant product line information. Also recall that many different types of artifacts (e.g., requirements, architecture, design models, source code) are used in the cases (cf. Section 5.2), and that each artifact type has its own technical representation of variation points, which challenges obtaining a global view of variability. This challenge is closely related to Challenges 1 and 2, requiring variability support in modeling languages, tool integrations, and traceability support.

Missing Standards Standardisation efforts in the direction to provide unified representations for variability and variation points exist, such as CVL (Haugen et al. 2013a) and VEL (Schulze and Hellebrand 2015). However, CVL was never adopted as a standard and VEL is explicitly aimed at providing an exchange language between tools rather than a fully integrated variability representation to be edited directly. Recently, a new initiative[3] was launched, aiming again at establishing a common feature modeling language, supporting common, community-agreed usage scenarios for such a language (Berger and Collet 2019b).

Representation of Behavioral Variability Furthermore, the cases from the literature CCT (Clements et al. 2001a), ENEA (Andersson and Bosch 2005), and the study of Chen and Ali Babar (2010) request the support for quality properties when representing variability. Specifically, "structural variability is well supported, behavioral and timing aspects are not" (Chen and Ali Babar 2010).

Representation of Topological Variability We observed the need for representing topological variability in the five cases that use design models (hardware modeling, modeling platform, chip modeling, aerospace, and power electronics). These are often XML-based domain-specific modeling languages (DSMLs) using graphs as their underlying structure. As opposed to feature-oriented (switch on/off features) variability, topological variability requires dedicated modeling languages. Typically, organizations create their own domain-specific languages (DSLs) for this reason, given limitations of established variability modeling languages (Berger et al. 2014d; Fantechi 2013; Behjati et al. 2014). A challenge is that topological variability can hardly be expressed using variability annotations and preprocessors, but typically requires more flexible code generators.

Representation of User-Interface Variability An interesting need was explained for the web application case, where the prime driver of variability is the customization of user interfaces. The current web frameworks do not offer any facilities for customization, so the company needs to rely on clone & own. UI variability has several peculiarities compared to traditional source code, as UIs encapsulate human computer interaction (HCI) assets (e.g., dialog models, context models, and aesthetic concerns). This challenge is related to Challenge 1 (MDE and code generation), given that abstract HCI models are sometimes

[3] https://modevar.github.io

used before the generation of concrete HCI implementations in target languages (Martinez et al. 2017). Also note that web applications as cases are rather rare among the existing literature. Exceptions are HomeAway (Krueger et al. 2008) and MarketMaker (Verlage and Kiesgen 2005); however, none of the publications describes the realization of variability in the web-based user interfaces of these web applications.

***Challenge 9,* Feature Modeling** In the light of obtaining a unified representation of variability, almost all of our subjects (9 of 12) strive to adopt feature modeling. For the power electronics case, a feature model would aim at visualizing the variant space and keeping the variants manageable. For the automotive firmware case, a feature model would be the basis for a configurator with intelligent configuration facilities, which should support product derivation using defaults, choice propagation, and conflict resolution—easing the configuration (and avoiding having to decide all features) Likewise, for the modeling platform case, an improved product derivation through an intelligent configurator tool is needed. In the imaging technology case, a more intuitive representation of configuration knowledge is needed, where even configuration profiles (partial configurations) or a feature hierarchy is seen as beneficial. In fact, as explained for the automotive firmware case, the feature hierarchy is considered the most valuable information. Furthermore, the experience report by Audi (Hardung et al. 2004) also requests modeling guidelines. A first step in this direction are probably the feature modeling principles of Nesic et al. (2019).

Feature-Modeling Process As such, a feature modeling process is needed, that supports obtaining a feature model from a diverse set of variability information sources (Bécan et al. 2016; Davril et al. 2013; Krueger et al. 2019a), integrated with feature-model management, merging, and synthesis techniques (Acher et al. 2010, 2013a, b, She et al. 2011, 2014). Our cases express that functional evolution is well reflected in commit messages (e.g., bug fix information, new feature implementations, new branches), which confirms some insights that commit messages are a good source for feature identification (Krueger et al. 2018a, 2019a; Zhou et al. 2018).

Variation-Point Methodology Another issue pertaining to such a process is a variation-point methodology (expressed for the aerospace case). It should guide, upon the decision to introduce a feature (originating from some scoping process), the flow of variation points into requirements, code, models, and other affected artifact types.

Feature-Oriented Authorization For the chip modeling, aerospace, and requirements engineering case, the need for user access control and authorization mechanisms was explicitly mentioned. Users have different access rights for different parts (features) of the product line. Especially the aerospace case needs to control visibility of variation points, where certain variation points or variants even need to be completely invisible (i.e., no trace should be visible that a variation point exists) for unauthorized personnel. Unfortunately, beyond work by Fægri and Hallsteinsen (2006), this challenge has not yet been recognized or received attention in the research community.

Feature-Oriented Views and Synchronization For nine of our twelve cases, the need for better visualizations, especially establishing feature-oriented views (Acher et al. 2011a; Andam et al. 2017; Montalvillo and Díaz 2015, 2017; Passos et al. 2013; Martinez et al. 2014), was expressed. In one of our cases it is

planned to reverse-engineer feature constraints from the codebase and distributed feature databases, including identifying the model hierarchy (Nadi et al. 2014, 2015), and to synthesize a feature model. This challenge is also reported by the exploratory study of Chen and Ali Babar (2010), and by the survey of Berger et al. (2013a).

An additional issue arising (related to Challenge 1 below) is that materialized views need to be continuously synchronized with the information sources, including feature databases, which cannot be abandoned. In other words, feature models as views degrade and need to be re-synthesized or synchronized continuously, as expressed for the automotive firmware case.

***Challenge 10,* Quality Assurance** Our cases expressed various challenges with respect to assuring the quality of their variants or platforms. While most responses were rather vague (e.g., the power electronics case: combine testing and SPLE to reduce efforts), since the cases strive to adopt more systematic variability management in the first place, they expressed challenges with respect to assuring the consistency of assets, more effective validation techniques, and the verification of platforms or many variants. Also note that dedicated studies on product-line analyses in industrial practice exist (Mukelabai et al. 2018b).

Consistency Analyses Recall that many cases strive to adopt feature models or have at least an informal, but structured feature model (cf. Table 2). When asked about analyses, they also confirmed that they would like to use feature-model analyses to assure their consistency, especially in relation to the other assets (including code and requirements with functional or quality properties), and under the continuous increase of concurrently maintained variants.

A further requested consistency-related analysis is dead-code analysis (automotive firmware case), and our requirements engineering case suggests a metrics-based approach (El-Sharkawy et al. 2019; Berger and Guo 2014a) for completeness, correctness, and consistency of requirements.

Validation and Verification The following two cases explicitly expressed the need for assuring properties for all possible variants (a.k.a., variability-aware analysis Midtgaard et al., 2014; Liebig et al. 2013).

The truck manufacturing case requests safety verification for the widest possible number of variants with a scalable compositional verification approach. Such a compositional verification should verify each lowest-level component individually, then reuse the results for verifying component combinations, showing that functional or quality properties are fulfilled for large sets of configurations that cannot be tested exhaustively. To this end, source code needs to be annotated with formal specifications, which in turn needs a clear picture of the product lines, especially configuration constraints. In the truck manufacturing case, this requires consolidating information scattered across various databases before.

For our chip modeling case, a challenge is to extend the modeling tool with verification capabilities for complex properties covering, for instance, market data to physical constraints concerning heat dissipation or power consumption. Then all variants should comply with the quality attribute specified for the platform (which is a model).

Given the static nature of variability-aware analyses, we can see future research direction in effectively combining validation and verification techniques, making both variability-aware for product lines.

7.5 Product line evolution and modernization

Once an integrated platform has been established and a certain maturity has been achieved in working with it, the focus shifts towards its modernization and evolution. Our modeling platform case constantly aims at a more systematic management of existing variability. The automotive firmware case, on the other hand, seeks to eliminate redundancy in an already very advanced product line. In the automotive firmware, hardware modeling, and modeling platform case, the developers strive to constantly identify new and changed feature constraints to enhance configuration processes and, therefore, improve the platform's maturity.

In general, better support for evolving software product lines is already requested in the literature, specifically in the studies of Chen et al. (2010, 2011) (especially evolution of variability models, including dependency management), the survey (Berger et al. 2013a) (model evolution), as well as for the cases Danfoss (Fogdal et al. 2016) (evolution of architectures and interfaces, and evolving towards a software ecosystem, explained shortly), Axis and Securitas (Bosch 1999a, b) (information distribution, asset version management, asset dependency management, and reuse of assets in different contexts), Nokia Networks (van der Linden et al. 2007) (backwards compatibility of assets, and quick changes in technology that need to be incorporated into the product line), Nokia Mobile Phones (van der Linden et al. 2007) (continuous architecture conformance checking), and Volvo Cars and Scania (Eklund and Gustavsson 2013; Gustavsson and Eklund 2010). The latter even explain: "We have not found any literature describing how product lines are maintained; most of those we found described the transition to a software product line and those challenges." Valuable guidelines are provided by Svahnberg and Bosch (1999), but thorough support is still needed. Our cases face the following challenges.

***Challenge 11,* Artifact Synchronization** The synchronization of different platform artifacts, including code and feature model, was emphasized for half of our cases (railway, aerospace, modeling platform, web application, automotive firmware, and chip modeling). A question that arises is how to keep the feature model in sync, for instance, whether incremental updates or a frequent re-synthesis should be done. Co-evolution scenarios in SPLE have been studied (Hellebrand et al. 2017; Schulze et al. 2016), but have not yet been adopted in industry.

The synchronization challenge is, not surprisingly, further complicated by Challenge 1 (MDE), when models constitute the primary variable artifacts. A specific challenge expressed for the chip modeling and modeling platform case is co-evolution of the variability-aware language of the model and the product line (i.e., the feature model).

Finally, for our aerospace case, the organization expressed the need for a better coordination among teams developing new features on branches (i.e., project management and integration management).

***Challenge 12,* End-User and After-Market Traceability** A challenge explicitly mentioned for two of the cases (aerospace, automotive firmware) is the need to trace customer adaptations on the after-market, as well as the general use of software assets. The experience reports by Danfoss (Fogdal et al. 2016), Nokia Networks (van der Linden et al. 2007), Naval Underwater Warfare Center (Cohen et al. 2002, 2004a, b), and PHILIPS Medical Systems (van der Linden et al. 2007), as well as the survey of Berger et al. (2013a), also mention this challenge. The former explains: "Frequency converters can collect a lot of data. [...] This

information can be used for predictive maintenance purposes, not only for the frequency converter but also for the machine." Fogdal et al. (2016)

Our cases explain that it needs to be clear which specific configurations the customers use in order to offer dedicated support and keep the software up-to-date. As such, organizations have a need to keep track of product variations that have been delivered to customers in their own portfolio. Of course, the question arises how to efficiently store and efficiently keep such customer configuration databases updated.

Addressing this challenge would require continuous measurement, perhaps building upon monitoring support (Rabiser et al. 2019), which needs to be extended and integrated with variability management concepts.

***Challenge 13*, Dynamic Product-Line Platform** An improvement goal expressed for our aerospace case is to conceive its next-generation product-line architecture, which specifically aims at adopting dynamic variability. This challenge is also pointed out in existing experience reports on Enea (Andersson and Bosch 2005) and on Danfoss (Fogdal et al. 2016).

The challenge is to adopt more modern variability mechanisms (supporting late dynamic binding) to support reconfiguration in the aircraft simulator environment. In fact, the organization expresses the need to adopt an ecosystem platform and strategy (to include contributions from suppliers as well as COTS components) with respective variability mechanisms (Berger et al. 2014c; Seidl et al. 2017). As such, the challenge is not only to migrate to an architecture supporting dynamic variability, but also foster inter-organizational reuse by allowing to extend variation points dynamically with third-party contributions.

8 Threats to validity

We now consider threats on the construct, internal, and external validity of our multiple-case study, as well as its reliability. We also discuss our mitigation strategies using the categorization by Yin (2003) as a guideline.

Construct Validity The initial data used in this paper is taken from semi-structured use case descriptions. Even though, the companies involved in the study refined these descriptions several times to harmonize them, these descriptions still show diversity in terms of level of detail and relevance. We addressed this threat by following up with the companies in semi-structured interviews and focus groups. This gives us high confidence in the data we collected and used for analysis and also helped us avoid any bias by triangulating the different data sources. To further avoid bias, we continuously discussed the material and ensured that each piece of information was interpreted by at least two of the authors. The additional member checking of the analyzed results confirms the correctness.

Not all subjects that participated in this study were familiar with the established terminology in the SPLE literature. That meant that we needed to map the terminology used in the use case description to the terminology used in this paper. To avoid any bias in this regard and limit the freedom of interpretation of the researchers, critical concepts were member checked before interacting with the companies.

Internal Validity To ensure that we could establish valid cause and effect relationships, we compared our findings across our cases and synthesized our results from the overall picture that emerged from a joint analysis, thus following the recommendations in Miles et al.

(1994). The individuals we interviewed were highly skilled engineers that are currently working with variability and have reflected on their work with variants. We also made extensive use of cross-checking between the different cases and different investigators to ensure that findings are correct (Yin 2003). In addition, during the focus groups and interviews with the case companies, we started the data analysis by preliminary explanation-building (Yin 2003) together with the experts from the companies, a process that was later completed by the researchers. The results were then submitted to the case companies for member checking and subsequent refinement.

External Validity It is the nature of a study that includes multiple cases that it offers better generalizability than studies of a single case. We followed the process of analytical generalization to abstract beyond the specificities of the individual cases and identified underlying issues that are generic enough to affect a large class of cases. The fact that we found several instances for almost all of the challenges we identified provides confidence that our findings are applicable beyond the twelve cases in this study. In addition, our triangulation with the results of our lightweight literature review enhances the validity of the challenges we identified and formulated.

We used a purposive sample of companies that have expertise in software product lines when selecting our cases. We argue that this is an advantage: the cases we included understand the need to manage variability and have encountered the challenges we report on first hand. Thus, we gathered a more detailed and nuanced picture than if we had included cases with less maturity. Our sample supports the aims of the paper which is to show the level of adoption and the challenges for variability management in cases with a certain maturity. Since we selected mature organizations, our cases are, however, skewed towards rather large companies. Only large organizations can maintain relatively large product lines and achieve a high level of sophistication in working with them. We would expect that smaller organizations with smaller product lines will not encounter all challenges mentioned here with the same severity, but that many of them will still encounter issues on a smaller scale.

In addition, the sample contains companies from five European countries. Geographic distribution is a relevant factor since organizational culture is determined by the country the organization resides in Lok and Crawford (2004) and organizational culture in turn affects product line adoption (Böckle et al. 2002). However, we cannot control for cultural differences in other continents. In principle, companies in South America, Asia, or elsewhere could behave differently and thus exhibit different challenges. On the other hand, our challenges are related to very technical aspects, mainly architecture and process in the BAPO framework (van der Linden 2002; Obbink et al. 2000), as opposed to business and organization, which indicates that similar engineering companies face the same or very similar challenges. Replicating our study to confirm or refute individual challenges would be valuable future work.

We also aimed for diversity in terms of the domains in which the organizations are active. By combining nine cases that rely on variability to develop their products and three tool providers, we also integrate two different perspectives on product-line engineering. In addition, we also combine six companies that create physical systems that include hardware and software (e.g., aerospace and power electronics) and six companies that create only software (e.g., modeling platform and imaging technology) of which three are tool providers. Systems engineering companies like the ones we surveyed represent a large share of the value creation and the R&D investment within the European Union (POTTERS and Nicola 2018). As such, we address a highly relevant market segment.

Reliability The research methodology has been jointly developed by the authors and refined over several iterations, often using feedback from the study participants. However, in order to use the resources made available to us by the case companies to maximum effect, we often used rather informal information channels with a fast response rate for clarifications, e.g., phone calls that were not recorded. That means that some of the findings are based on notes by an individual researcher and not on mechanical recordings. We have addressed this by triangulation with more formal techniques and by member checking. In addition, we have shared all collected information between all involved researchers. We have also recorded all steps we took in a case study protocol (Yin 2003), an abbreviated version of which is presented in Section 3.

9 Conclusion

We presented a multiple-case study on the adoption and challenges of systematic variability management in industrial practice, organized in a common framework. Our twelve cases comprised nine use cases, where SPLE concepts are adopted for developing an organization's systems, and three tool cases, representing modeling tools for software and hardware, where the vendor sees a need for adopting SPLE concepts in the tools—enhancing variant-engineering practices among customers. Given that most experience reports (van der Linden et al. 2007) are more than a decade old and very diverse in terms of details reported, we provided an up-to-date snapshot on as many as twelve cases. With a specific focus on the adoption of concepts organized in a common framework, our work is also the first that systematically elicited this state of adoption from a range of companies—a surprisingly challenging task, given that we needed many iterations to understand the development and thereby also which concepts are used.

Variability Drivers Not surprisingly, hardware is still one of the most significant drivers, directly followed by market pressure for customization. What appears to gain increasing relevance are Industry 4.0 and digitalization, challenging variability management even more.

Adoption of SPLE Concepts Not surprisingly, our subjects from the automotive domain are most advanced in terms of adopting SPLE concepts. This highlights the importance of existing experience reports from this domain (Flores et al. 2012; Tischer et al. 2011; Gustavsson and Eklund 2010; Hardung et al. 2004; Dziobek et al. 2008; Thiel et al. 2001; Bayer et al. 2006). Yet, what this domain is still lacking is a well-unified management of variability, given a diversity of different asset types we observed, including features, configuration options, calibration parameters, components, and coarse-grained packages. All cases understand the benefit of features and strive to adopt them. It is a bit surprising that the concept of features is not well adopted. Perhaps, lifting programmers' work practices from low-level components to the domain-level is more challenging than expected. In fact, developers are not used to working on different abstraction levels (Berger et al. 2014b). Our automotive cases are a bit closer to adopting features, given the complexity of their variability. Yet, they do not use feature models or common SPLE tooling.

We observed a preliminary adoption of off-the-shelf SPLE tools in the domain, but since the adoption is still limited, the needs and requirements in this domain seem to not yet been fully understood and met.

While we omitted detailed information about the case companies for anonymity reasons, we can see an increased adoption of concepts in the large cases. Yet, our study again shows the need for adoption also in smaller companies, including those in the web application domain. Experience reports and surveys on adoption in such smaller companies (Verlage and Kiesgen 2005; Thörn and Gustafsson 2008, 2010) and strategies for incremental adoption of concepts (Antkiewicz et al. 2014) are certainly relevant to consider in future research.

Challenges We observe that the adoption of SPLE concepts is still a tool-integration problem, given all the different types of artifacts and existing tooling, which engineers are familiar with and that is core to the development. How this situation can be improved, and whether there are research questions (as opposed to it being just an engineering problem that tool integrators need to solve), are interesting questions to be answered in the future. Some steps in this direction were taken with the publicly available Common Variability Language proposal (Haugen et al. 2013b), the Variability Exchange Language (VEL 2018), the recent initiative for a common feature-modeling language (cf. Challenge 8 in Section 7.4), or a recent Dagstuhl seminar on unifying versioning and variability concepts (Berger et al. 2019a). However, more agreement and adoption will be needed to consider it. Among the most requested needs by our practitioners are a better representation and visualization of variability, process support for conducting the migration, support for continuous integration, and traceability throughout the lifecycle including the aftermarket. Surprisingly, product-line analysis for validation or verification played a rather minor role in the results of our literature review and among the challenges faced by our cases, perhaps since more systematic variability management needs to be adopted in the first place.

Future Work Finally, beyond the scope of this study, it would be valuable future work to identify and justify research directions that are addressed in the research community, but that are not worth investigating based on empirical evidence. Such studies probably would need to focus on particular sub-directions in product-line research, and might be hotly contested among the community, but would apparently be very useful to scope future research to focus on relevant challenges.

Acknowledgments We thank all practitioners who participated in the focus group discussions, as well as we thank the anonymous reviewers for very valuable comments to improve the paper. This work is supported by Vinnova Sweden, Fond Unique Interministériel (FUI) France, and the Swedish Research Council.

Funding Information Open access funding provided by University of Gothenburg.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

Abukwaik H, Burger A, Andam B, Berger T (2018) Semi-automated feature traceability with embedded annotations. In: ICSME, NIER Track

Acher M, Collet P, Lahire P, France R (2010) Composing feature models. In: Proceedings of the Second International Conference on Software Language Engineering, SLE'09

Acher M, Collet P, Lahire P, France R (2011a) Slicing feature models. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering. IEEE Computer Society

Acher M, Cleve A, Collet P, Merle P, Duchien L, Lahire P (2011b) Reverse engineering architectural feature models. In: Proceedings of the 5th European Conference on Software Architecture, ECSA'11

Acher M, Collet P, Lahire P, France R (2013a) FAMILIAR:, A domain-specific language for large scale management of feature models. Sci Comput Program 78(6):657–681. https://doi.org/10.1016/j.scico.2012.12.004

Acher M, Baudry B, Heymans P, Cleve A, Hainaut JL (2013b) Support for reverse engineering and maintaining feature models. In: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, pp 20

Akesson J, Nilsson S, Krüger J, Berger T (2019) Migrating the android apo-games into an annotation-based software product line. In: SPLC

Ali MS, Babar MA, Schmid K (2009) A comparative survey of economic models for software product lines. In: 35Th euromicro conference on software engineering and advanced applications, SEAA 2009, proceedings. IEEE Computer Society, Patras, pp 275–278. https://doi.org/10.1109/SEAA.2009.89

Andam B, Burger A, Berger T, Chaudron M (2017) FLOrIDA: Feature LOcation DAshboard for Extracting and Visualizing Feature Traces. In: Vamos

Andersson J, Bosch J (2005) Development and use of dynamic product-line architectures. IEE Proc-Softw 152(1):15–28

Antkiewicz M, Ji W, Berger T, Czarnecki K, Schmorleiz T, Lämmel R., Stanciulescu S, Wasowski A, Schäfer I (2014) Flexible product line engineering with a virtual platform. In: ICSE

Apel S, Kästner C (2009) An overview of feature-oriented software development. J Obj Technol 8(5):49–84

Apel S, Batory D, Kästner C, Saake G (2013) Feature- oriented software product lines. Springer, Berlin

Assunção WKG, Lopez-Herrejon RE, Linsbauer L, Vergilio SR, Egyed A (2017) Reengineering legacy applications into software product lines: a systematic mapping. Empir Softw Eng 22(6):2972–3016

Atzori L, Iera A, Morabito G (2010) The internet of things: a survey. Comput Netw 54(15):2787–2805

Bartholdt J, Becker D (2011) Re-engineering of a hierarchical product line. In: Proceedings of the 2011 15th International Software Product Line Conference, SPLC '11. IEEE Computer Society, Washington, pp 232–240. https://doi.org/10.1109/SPLC.2011.16

Bass L, Clements P, Kazman R (2003) Software architecture in practice. Addison-Wesley Professional

Bastos JF, da Mota Silveira Neto PA, O'Leary P, de Almeida ES, de Lemos Meira SR (2017) Software product lines adoption in small organizations. J Syst Softw 131(Supplement C):112 – 128. https://doi.org/10.1016/j.jss.2017.05.052. http://www.sciencedirect.com/science/article/pii/S0164121217300997

Bayer J, Forster T, Lehner T, Giese C, Schnieders A, Weiland J (2006) Process family engineering in automotive control systems: a case study. In: GPCE

Bécan G, Acher M, Baudry B, Nasr SB (2016) Breathing ontological knowledge into feature model synthesis: an empirical study. Empir Softw Eng 21(4):1794–1841

Behjati R, Nejati S, Briand LC (2014) Architecture- level configuration of large-scale embedded software systems. ACM Trans Softw Eng Methodol 23(3):25:1–25:43

Behringer B, Palz J, Berger T (2017) Peopl: Projectional editing of product lines. In: 2017 IEEE/ACM 39Th international conference on software engineering (ICSE), pp 563–574. https://doi.org/10.1109/ICSE.2017.58

Benavides D, Segura S, Ruiz-Cortés A (2010) Automated analysis of feature models 20 years later: a literature review. Inf Syst 35(6):615–636

Berger T, She S, Lotufo R, Czarnecki K, Wasowski A (2010a) Feature-to-code mapping in two large product lines. In: SPLC

Berger T, She S, Lotufo R, Wasowski A, Czarnecki K (2010b) Variability modeling in the real: a perspective from the operating systems domain. In: ASE

Berger T, Rublack R, Nair D, Atlee JM, Becker M, Czarnecki K, Wasowski A (2013a) A survey of variability modeling in industrial practice. In: Vamos

Berger T, She S, Lotufo R, Wasowski A, Czarnecki K (2013b) A study of variability models and languages in the systems software domain. IEEE Trans Softw Eng 39(12):1611–1640

Berger T, Guo J (2014a) Towards system analysis with variability model metrics. In: Vamos

Berger T, Nair D, Rublack R, Atlee JM, Czarnecki K, Wasowski A (2014b) Three cases of feature-based variability modeling in industry. In: MODELS
Berger T, Pfeiffer RH, Tartler R, Dienst S, Czarnecki K, Wasowski A, She S (2014c) Variability Mechanisms in Software Ecosystems. Inf Softw Technol 56(11):1520–1535
Berger T, Stanciulescu S, Ogaard O, Haugen O, Larsen B, Wasowski A (2014d) To connect or not to connect: Experiences from modeling topological variability. In: SPLC
Berger T, Lettner D, Rubin J, Grünbacher P, Silva A, Becker M, Chechik M, Czarnecki K (2015) What is a feature? A qualitative study of features in industrial software product lines. In: SPLC
Berger T, Chechik M, Kehrer T, Wimmer M (2019a) Software evolution in time and space: Unifying version and variability management (dagstuhl seminar 19191). In: Dagstuhl reports. Schloss dagstuhl – leibniz-zentrum fuer informatik
Berger T, Collet P (2019b) Usage scenarios for a common feature modeling language. In: First international workshop on languages for modelling variability (MODEVAR)
Bergey J, Cohen S, Jones L, Smith D (2004) Software product lines: Experiences from the sixth dod software product line workshop. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst
Beuche D (2004) pure::variants Eclipse Plugin. User Guide. pure-systems GmbH. Available from http://web.pure-systems.com/fileadmin/downloads/pv_userguide.pdf
Böckle G, Munoz JB, Knauber P, Krueger C, do Prado Leite, JCS, van der Linden F, Northrop L, Stark M, Weiss DM (2002) Adopting and institutionalizing a product line culture. In: Chastek GJ (ed) Software product lines. Springer, Berlin, pp 49–59
Bosch J (1999a) Evolution and composition of reusable assets in product-line architectures: A case study. In: Proceedings of the TC2 First Working IFIP Conference on Software Architecture (WICSA1), WICSA1. Kluwer, B.V., Deventer, The Netherlands, pp 321–340. http://dl.acm.org/citation.cfm?id=646545.696377
Bosch J (1999b) Product-line architectures in industry: a case study. In: Proceedings of the 21st International Conference on Software Engineering, ICSE '99. ACM, New York, pp 544–554. https://doi.org/10.1145/302405.302690
Bosch J (2000) Design and Use of Software Architectures: Adopting and Evolving a Product-line Approach. ACM Press/Addison-Wesley Publishing Co., New York
Bosch J (2002) Maturity and evolution in software product lines: approaches, artefacts and organization. In: SPLC
Brownsword L, Clements P (1996) A case study in successful product line development. Technical report, Software Engineering Institute Carnegie Mellon University
Buhrdorf R, Churchett D, Krueger C (2003) Salion's experience with a reactive software product line approach. In: International workshop on software product-family engineering. Springer, pp 317–322
Businge J, Moses O, Nadi S, Bainomugisha E, Berger T (2018) Clone-based variability management in the android ecosystem. In: ICSME
Chastek G, Donohoe P, McGregor JD, Muthig D (2011) Engineering a production method for a software product line. In: Proceedings of the 2011 15th International Software Product Line Conference, SPLC '11. IEEE Computer Society, Washington, pp 277–286. https://doi.org/10.1109/SPLC.2011.46
Clements P, Cohen S, Donohoe P, Northrop L (2001a) Control channel toolkit: a software product line case study. Technical report, Software Engineering Institute Carnegie Mellon University
Clements P, Northrop L (2001b) Software product lines: Practices and patterns. Addison-Wesley
Clements P, Northrop LM (2002) Salion, inc.: a software product line case study. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst
Clements P, Bergey J (2005) The us army's common avionics architecture system (caas) product line: A case study. Technical report, Software Engineering Institute Carnegie Mellon University
Chen L, Ali Babar M, Ali N (2009) Variability management in software product lines: a systematic review. In: SPLC'09
Chen L, Ali Babar M (2010) Variability management in software product lines: an investigation of contemporary industrial challenges. In: SPLC'10
Chen L, Babar MA (2011) A systematic review of evaluation of variability management approaches in software product lines. Inf Softw Technol 53(4):344–362
Cohen S, Dunn E, Soule A (2002) Successful product line development and sustainment: a dod case study. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst
Cohen S, Zubrow D, Dunn E (2004a) Acquisition pilot: Product line acquisition and measurement at nuwc. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst
Cohen S, Zubrow D, Dunn E (2004b) Case study: a measurement program for product lines. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst

Cohen MB, Dwyer MB, Shi J (2007) Interaction testing of highly-configurable systems in the presence of constraints. In: ISSTA

Czarnecki K, Eisenecker UW (2000) Generative programming: methods, Tools, and Applications. Addison-Wesley, Boston

Czarnecki K, Grünbacher P, Rabiser R, Schmid K, Wasowski A (2012) Cool features and tough decisions: a comparison of variability modeling approaches. In: Vamos

Davril JM, Delfosse E, Hariri N, Acher M, Cleland-Huang J, Heymans P (2013) Feature model extraction from large collections of informal product descriptions. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering. ACM, pp 290–300

Debbiche J, Lignell O, Krüger J, Berger T (2019) Migrating the java-based apo-games into a composition-based software product line. In: SPLC

Dietrich C, Tartler R, Schröder-Preikschat W, Lohmann D (2012) A robust approach for variability extraction from the linux build system. In: Proceedings of the 16th International Software Product Line Conference - Volume 1, SPLC '12. ACM, New York, pp 21–30. https://doi.org/10.1145/2362536.2362544

Dikel D, Kane D, Ornburn S, Loftus W, Wilson J (1997) Applying software product-line architecture. Computer 30(8):49–55. https://doi.org/10.1109/2.607064

Dordowsky F, Hipp W (2009) Adopting software product line principles to manage software variants in a complex avionics system. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09. Carnegie Mellon University, Pittsburgh, pp 265–274. http://dl.acm.org/citation.cfm?id=1753235.1753272

Dubinsky Y, Rubin J, Berger T, Duszynski S, Becker M, Czarnecki K (2013) An exploratory study of cloning in industrial software product lines. In: CSMR

Duc AN, Mockus A, Hackbarth R, Palframan J (2014) Forking and coordination in multi-platform development: a case study. In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14

Dziobek C, Loew J, Przystas W, Weiland J (2008) Functional variants handling in simulink models. In: MACDE. https://www.researchgate.net/publication/238778580_Functional_Variants_Handling_in_Simulink_Models

Eklund U, Gustavsson H (2013) Architecting automotive product lines: Industrial practice. Sci Comput Program 78(12):2347–2359

El-Sharkawy S, Yamagishi-Eichler N, Schmid K (2019) Metrics for analyzing variability and its implementation in software product lines: a systematic literature review. Inf Softw Technol 106:1–30

Entekhabi S, Solback A, Steghöfer JP, Berger T (2019) Visualization of feature locations with the tool featuredashboard. In: 23Rd international systems and software product line conference (SPLC), tools track

Etikan I, Musa SA, Alkassim RS (2016) Comparison of convenience sampling and purposive sampling. Amer J Theor Appl Stat 5(1):1–4

Fægri TE, Hallsteinsen S (2006) A software product line reference architecture for security. In: Software product lines. Springer, pp 275–326

Fantechi A (2013) Topologically configurable systems as product families. In: SPLC

Faulk SR (2001) Product-line requirements specification (prs): an approach and case study. In: Proceedings of the Fifth IEEE International Symposium on Requirements Engineering, RE '01. IEEE Computer Society, Washington

Faust D, Verhoef C (2003a) Software product line migration and deployment. Softw Pract Exper 33(10):933–955

Faust D, Verhoef C (2003b) Software Product Line Migration and Deployment. Softw: Pract Exper 33(10):933–955

Fenske W, Thüm T, Saake G (2014) A taxonomy of software product line reengineering. In: Vamos

Fischer S, Linsbauer L, Lopez-herrejon RE, Egyed A (2014) Enhancing clone-and-own with systematic reuse for developing software variants. In: 30Th IEEE international conference on software maintenance and evolution, victoria, BC. IEEE Computer Society, Canada, pp 391–400. https://doi.org/10.1109/ICSME.2014.61

Flores R, Krueger C, Clements P (2012) Mega-Scale product line engineering at general motors. In: Proceedings of SPLC

Fogdal T, Scherrebeck H, Kuusela J, Becker M, Zhang B (2016) Ten years of product line engineering at danfoss: lessons learned and way ahead. In: SPLC

Fowler M (1999) Refactoring: Improving the design of existing code. Addison-wesley Longman Publishing co., Inc., Boston

Ganesan D, Lindvall M, Ackermann C, McComas D, Bartholomew M (2009) Verifying architectural design rules of the flight software product line. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09. Carnegie Mellon University, Pittsburgh, pp 161–170

Gannod GC, Lutz RR, Cantu M (2001) Embedded software for a space interferometry system: automated analysis of a software product line architecture. In: Conference Proceedings of the 2001 IEEE International Performance, Computing, and Communications Conference. IEEE, pp 145–150

Ganz C, Layes M (1998) Modular turbine control software: a control software architecture for the abb gas turbine family. In: International workshop on architectural reasoning for embedded systems

Garcia S, Strueber D, Brugali D, Fava AD, Schillinger P, Pelliccione P, Berger T (2019) Variability modeling of service robots: Experiences and challenges. In: 13Th international workshop on variability modelling of software-intensive systems (vamos)

Gustavsson H, Eklund U (2010) Architecting automotive product lines: Industrial practice. In: SPLC

Habli I, Kelly T (2007) Challenges of establishing a software product line for an aerospace engine monitoring system. In: Proceedings of the 11th International Software Product Line Conference, SPLC '07. IEEE Computer Society, Washington, pp 193–202. https://doi.org/10.1109/SPLC.2007.14

Hardung B, Kölzow T, Krüger A (2004) Reuse of software in distributed embedded automotive systems. In: Proceedings of the 4th ACM International Conference on Embedded Software, EMSOFT '04

Haugen Ø, Wasowski A, Czarnecki K (2013a) CVL: common variability language. In: Kishi T, Jarzabek S, Gnesi S (eds) 17th International Software Product Line Conference, SPLC 2013. ACM, Tokyo, p 277. https://doi.org/10.1145/2491627.2493899

Haugen O, Wasowski A, Czarnecki K (2013b) Cvl: Common variability language. In: Proceedings of the 17th International Software Product Line Conference, SPLC '13. ACM, New York, pp 277–277. https://doi.org/10.1145/2491627.2493899

Hellebrand R, Schulze M, Ryssel U (2017) Reverse engineering challenges of the feedback scenario in co-evolving product lines. In: Ter Beek MH, Cazzola W, Diaz O, Rosa ML, Lopez-Herrejon RE, Thüm T, Troya J, Cortés AR, Benavides D (eds) Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017, vol B. ACM, Sevilla, pp 53–56. https://doi.org/10.1145/3109729.3109735

Hess KD, Dordowsky F (2008) Rational clearcase migration to a complex avionics project - an experience report. In: CONQUEST

Hetrick WA, Krueger C, Moore JG (2006) Incremental return on incremental investment: Engenio's transition to software product line practice. In: Companion to the 21st ACM SIGPLAN symposium on object-oriented programming systems, languages, and applications, OOPSLA '06. ACM, New York, pp 798–804. https://doi.org/10.1145/1176617.1176726

Jensen P (2007a) Experiences with product line development of multi-discipline analysis software at overwatch textron systems. In: Proceedings of the 11th International Software Product Line Conference, SPLC '07. IEEE Computer Society, Washington, pp 35–43. https://doi.org/10.1109/SPLC.2007.18

Jepsen HP, Dall JG, Beuche D (2007b) Minimally invasive migration to software product lines. In: SPLC

Jepsen HP, Beuche D (2009) Running a software product line: standing still is going backwards. In: SPLC

Ji W, Berger T, Antkiewicz M, Czarnecki K (2015) Maintaining Feature Traceability with Embedded Annotations. In: SPLC

Johansen MF, Haugen Ø, Fleurey F, Carlson E, Endresen J, Wien T (2012) A technique for agile and automatic interaction testing for product lines. In: Nielsen B, Weise C (eds) Testing Software and Systems - 24th IFIP WG 6.1 International Conference, ICTSS 2012, Proceedings, Lecture Notes in Computer Science, vol 7641. Springer, Aalborg, pp 39–54. https://doi.org/10.1007/978-3-642-34691-0_5

John I, Knauber P, Muthig D, Widen T (2001) Qualifikation von kleinen und mittleren unternehmen (kmu) im bereich software variantenbildung. Technical report IESE-026.00/D, Fraunhofer IESE

Kang K, Cohen S, Hess J, Nowak W, Peterson S (1990) Feature-oriented domain analysis (FODA) feasibility study. Technical Report, SEI CMU

Kästner C, Thum T, Saake G, Feigenspan J, Leich T, Wielgorz F, Apel S (2009) featureIDE: A Tool Framework for Feature-oriented Software Development. In: Proceedings of ICSE'09

Kästner C, Ostermann K, Erdweg S (2012) A variability-aware module system. In: G.T. Leavens, M.B. Dwyer (eds.) Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012. ACM, Tucson, pp 773–792. https://doi.org/10.1145/2384616.2384673

Kitchenham B, Pfleeger SL (2002) Principles of survey research: parts 1–6. ACM SIGSOFT Software Engineering Notes 26–28 (2001–2003)

Kolassa C, Rendel H, Rumpe B (2015) Evaluation of variability concepts for simulink in the automotive domain. In: Bui TX, Louis Jr RHS (eds) 48th Hawaii International Conference on System Sciences, HICSS 2015. IEEE Computer Society, Kauai, pp 5373–5382. https://doi.org/10.1109/HICSS.2015.632

Kolb R, Muthig D, Patzke T, Yamauchi K (2005) A case study in refactoring a legacy component for reuse in a product line. In: Proceedings of the 21st IEEE International Conference on Software Maintenance, ICSM '05. IEEE Computer Society, Washington, pp 369–378. https://doi.org/10.1109/ICSM.2005.5

Krueger C (2006) New methods in software product line development. In: Software product lines, 10th international conference, SPLC 2006, Proceedings. IEEE Computer Society, Baltimore, pp 95–102. https://doi.org/10.1109/SPLINE.2006.1691581

Krueger C (2007) BigLever Software Gears and the 3-tiered SPL Methodology. In: Proceedings of OOPSLA'07 companion

Krueger C, Churchett D, Buhrdorf R (2008) Homeaway's transition to software product line practice: Engineering and business results in 60 days. In: Proceedings of the 2008 12th International Software Product Line Conference, SPLC '08. IEEE Computer Society, Washington, pp 297–306. https://doi.org/10.1109/SPLC.2008.36

Krüger J, Fenske W, Meinicke J, Leich T, Saake G (2016) Extracting software product lines: a cost estimation perspective. In: Proceedings of the 20th International Systems and Software Product Line Conference, SPLC '16. ACM, New York, pp 354–361. https://doi.org/10.1145/2934466.2962731

Krüger J, Nielebock S, Krieter S, Diedrich C, Leich T, Saake G, Zug S, Ortmeier F (2017) Beyond software product lines: Variability modeling in cyber-physical systems. In: SPLC

Krueger J, Gu W, Shen H, Mukelabai M, Hebig R, Berger T (2018a) Towards a better understanding of software features and their characteristics: a case study of marlin. In: Vamos

Krüger J, Berger T, Leich T (2018b) Features and how to find them: a survey of manual feature location. LLC/CRC Press

Krueger J, Mukelabai M, Gu W, Shen H, Hebig R, Berger T (2019a) Where is my feature and what is it about? a case study on recovering feature facets Journal of Systems and Software

Krueger J, Calikli G, Berger T, Leich T, Saake G (2019b) Effects of explicit feature traceability on program comprehension. In: 27Th ACM SIGSOFT international symposium on the foundations of software engineering (FSE)

Lanman J, Darbin R, Rivera J, Clements P, Krueger C (2013) The challenges of applying service orientation to the u.s. army's live training software product line. In: Proceedings of the 17th International Software Product Line Conference, SPLC '13. ACM, New York, pp 244–253. https://doi.org/10.1145/2491627.2491649

Li D, Chang CK (2009) Initiating and institutionalizing software product line engineering: From bottom-up approach to top-down practice. In: Proceedings of the 2009 33rd Annual IEEE International Computer Software and Applications Conference - Volume 01, COMPSAC '09. IEEE Computer Society, Washington, pp 53–60. https://doi.org/10.1109/COMPSAC.2009.17

Liang L, Hu Z, Wang X (2005) An open architecture for medical image workstation. In: Medical imaging 2005: PACS and imaging informatics

Liebig J, von Rhein A, Kästner C, Apel S, Dörre J, Lengauer C (2013) Scalable analysis of variable software. In: ESEC/FSE

Lillack M, Stanciulescu S, Hedman W, Berger T, Wasowski A (2019) Intention-based integration of software variants. In: 41St international conference on software engineering, ICSE

Linåker J, Sulaman S, Maiani de Mello R, Höst M (2015) Guidelines for Conducting Surveys in Software Engineering. Lund University. https://portal.research.lu.se/portal/files/6062997/5463412.pdf

van der Linden F (2002) Software product families in europe: The esaps & café projects. IEEE Softw 19(4):41–49

van der Linden FJ, Schmid K, Rommes E (2007) Software product lines in action: The best industrial practice in product line engineering. Springer, Berlin

Linsbauer L, Berger T, Grünbacher P (2017) A classification of variation control systems. In: Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences, GPCE 2017. ACM, New York, pp 49–62

Lok P, Crawford J (2004) The effect of organisational culture and leadership style on job satisfaction and organisational commitment: a cross-national comparison. J Manag Develop 23(4):321–338

Marimuthu C, Chandrasekaran K (2017) Systematic studies in software product lines: a tertiary study. In: 21St international systems and software product line conference - Volume A, SPLC '17

Martinez J, Ziadi T, Mazo R, Bissyandé TF, Klein J, Traon YL (2014) Feature relations graphs: A visualisation paradigm for feature constraints in software product lines. In: H.a. sahraoui, A. Zaidman, B. Sharif (eds.) Second IEEE Working Conference on Software Visualization, VISSOFT 2014. IEEE Computer Society, Victoria, pp 50–59. https://doi.org/10.1109/VISSOFT.2014.18

Martinez J, Ziadi T, Bissyandé TF, Klein J, Traon YL (2015) Bottom-up adoption of software product lines: a generic and extensible approach. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015. ACM, Nashville, pp 101–110

Martinez J, Sottet J, García Frey A, Ziadi T, Bissyandé TF, Vanderdonckt J, Klein J, Traon YL (2017) Variability management and assessment for user interface design. In: Sottet, J, García Frey, A, Vanderdonckt, J (eds) Human Centered Software Product Lines, Human-Computer Interaction Series. Springer, pp 81–106. https://doi.org/10.1007/978-3-319-60947-8_3

Matsumoto Y (2007) A guide for management and financial controls of product lines. In: 11Th international software product line conference (SPLC 2007), pp 163–170. https://doi.org/10.1109/SPLINE.2007.26

Melo J, Brabrand C, Wasowski A (2016) How does the degree of variability affect bug finding?. In: International conference on software engineering (ICSE)

Midtgaard J, Brabrand C, Wasowski A (2014) Systematic derivation of static analyses for software product lines. In: MODULARITY

Miles MB, Huberman AM, Huberman M, Huberman M (1994) Qualitative data analysis: an expanded sourcebook. Sage, New York

Mohagheghi P, Conradi R (2007) Quality, productivity and economic benefits of software reuse: a review of industrial studies. Empir Softw Eng 12(5):471–516

Mohagheghi P, Conradi R (2008) An empirical investigation of software reuse benefits in a large telecom product. ACM Trans Softw Eng Methodol 17(3):13:1–13:31. https://doi.org/10.1145/1363102.1363104

Molléri JS, Petersen K, Mendes E (2016) Survey guidelines in software engineering: an annotated review. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '16. ACM, New York, pp 58:1–58:6. https://doi.org/10.1145/2961111.2962619

Montalvillo L, Díaz O (2015) Tuning github for spl development: Branching models & repository operations for product engineers. In: SPLC

Montalvillo L, Díaz O, Azanza M (2017) Visualizing product customization efforts for spotting spl reuse opportunities. In: SPLC

Mukelabai M, Behringer B, Fey M, Palz J, Krüger J, Berger T (2018a) Multi-view editing of software product lines with peopl. In: 40Th international conference on software engineering (ICSE), demonstrations track

Mukelabai M, Nesic D, Maro S, Berger T, Steghöfer JP (2018b) Tackling combinatorial explosion: a study of industrial needs and practices for analyzing highly configurable systems. In: 33Rd IEEE/ACM international conference on automated software engineering (ASE)

Nadi S, Berger T, Kästner C., Czarnecki K (2014) Mining configuration constraints: Static analyses and empirical results. In: ICSE

Nadi S, Berger T, Kästner C, Czarnecki K (2015) Where do configuration constraints stem from? an extraction approach and an empirical study. IEEE Transactions on Software Engineering. Preprint

Nesic D, Krueger J, Stanciulescu S, Berger T (2019) Principles of feature modeling. In: FSE

Obbink H, Müller J, America P, van Ommering R, Muller G, van der Sterren W, Wijnstra J (2000) COPA: a component-oriented platform architecting method for families of software-intensive electronic products. Tutorial for SPLC

Passos L, Czarnecki K, Apel S, Wasowski A, Kästner C, Guo J (2013) Feature-oriented software evolution. In: Vamos

Passos L, Padilla J, Berger T, Apel S, Czarnecki K, Valente MT (2015) Feature scattering in the large: a longitudinal study of linux kernel device drivers. In: MODULARITY

Passos L, Queiroz R, Mukelabai M, Berger T, Apel S, Czarnecki K, Padilla J (2018) A study of feature scattering in the linux kernel. IEEE Transactions on Software Engineering, Preprint

Pech D, Knodel J, Carbon R, Schitter C, Hein D (2009) Variability management in small development organizations: Experiences and lessons learned from a case study. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09. Carnegie Mellon University, Pittsburgh, pp 285–294

Perrouin G, Sen S, Klein J, Baudry B, Le Traon Y (2010) Automated and scalable t-wise test case generation strategies for software product lines. In: ICST

Pfofe T, Thüm T, Schulze S, Fenske W, Schaefer I (2016) Synchronizing software variants with variantsync. In: H. Mei (ed.) Proceedings of the 20th International Systems and Software Product Line Conference, SPLC 2016. ACM, Beijing, pp 329–332. https://doi.org/10.1145/2934466.2962726

Pohjalainen P (2011) Bottom-up modeling for a software product line: an experience report on agile modeling of governmental mobile networks. In: Proceedings of the 2011 15th International Software Product Line Conference, SPLC'11

Pohl K, Böckle G, Linden FJvd (2005) Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin

POTTERS L, Nicola G (2018) The 2018 EU Survey on Industrial R&D Investment Trends. Technical report. https://doi.org/10.2760/802408x

Quilty G, Cinneide MO (2011) Experiences with software product line development in risk management software. In: Proceedings of the 2011 15th International Software Product Line Conference, SPLC '11. IEEE Computer Society, Washington, pp 251–260. https://doi.org/10.1109/SPLC.2011.30

Rabiser R, Schmid K, Eichelberger H, Vierhauser M, Guinea S, Grünbacher P (2019) A domain analysis of resource and requirements monitoring: Towards a comprehensive model of the software monitoring domain. Inf Softw Technol 111:86–109

Rauf R, Antkiewicz M, Czarnecki K (2011) Logical structure extraction from software requirements documents. In: Proceedings of the 2011 IEEE 19th International Requirements Engineering Conference, RE '11

Romero D, Quinton C, Duchien L, Seinturier L, Valdez C (2015) Smartyco: Managing cyber-physical systems for smart environments. In: European conference on software architecture. Springer, pp 294–302

Rösel A (1998) Experiences with the evolution of an application family architecture. In: Proceedings of the Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families

Rosenmüller M, Siegmund N (2010) Automating the configuration of multi software product lines. In: Benavides, D, Batory, DS, Grünbacher, P (eds) Fourth International Workshop on Variability Modelling of Software-Intensive Systems, Linz, Austria, January 27-29, 2010. Proceedings, ICB-Research Report. Universität Duisburg-Essen, vol 37, pp 123–130. http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf

Rubin J, Chechik M (2013a) A framework for managing cloned product variants. In: ICSE

Rubin J, Czarnecki K, Chechik M (2013b) Managing cloned variants: a framework and experience. In: SPLC

Rubin J, Kirshin A, Botterweck G, Chechik M (2012) Managing forked product variants. In: SPLC

Sattler F, von Rhein A, Berger T, Johansson NS, Hardø MM, Apel S (2018) Lifting inter-app data-flow analysis to large app sets. Autom Softw Eng 25(2):315–346

Schaefer I, Bettini L, Bono V, Damiani F, Tanzarella N (2010) Delta-oriented programming of software product lines. In: International conference on software product lines. Springer, pp 77–91

Schmid K, John I, Kolb R, Meier G (2005) Introducing the pulse approach to an embedded system population at testo ag. In: ICSE

Schulze M, Hellebrand R (2015) Variability exchange language - A generic exchange format for variability data. In: Zimmermann, W, Böhm, W, Grelck, C, Heinrich, R, Jung, R, Konersmann, M, Schlaefer, A, Schmieders, E, Schupp, S, y Widemann, BT, Weyer, T (eds) Gemeinsamer Tagungsband der Workshops der Tagung Software Engineering 2015, CEUR Workshop Proceedings, vol 1337. CEUR-WS.org, Dresden, pp 71–80. http://ceur-ws.org/Vol-1337/paper11.pdf

Schulze S, Schulze M, Ryssel U, Seidl C (2016) Aligning coevolving artifacts between software product lines and products. In: I. Schaefer, V. Alves, E.S. de Almeida (eds.) Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems. ACM, Salvador, pp 9–16. https://doi.org/10.1145/2866614.2866616

Seidl C, Berger T, Elsner C, Schultis KB (2017) Challenges and solutions for opening small and medium-scale industrial sofware platforms. In: 21St international systems and software product line conference (SPLC)

Seiler M, Paech B (2017) Using tags to support feature management across issue tracking systems and version control systems. In: REFSQ

Sellier D, Benguria G, Urchegui G (2007) Introducing software product line engineering for metal processing lines in a small to medium enterprise. In: Proceedings of the 11th International Software Product Line Conference, SPLC '07. IEEE Computer Society, Washington, pp 54–62. https://doi.org/10.1109/SPLC.2007.22

Sharp DC (1998) Reducing avionics software cost through component based product line development. In: 17th DASC. AIAA/IEEE/SAE. Digital Avionics Systems Conference. Proceedings (Cat. No. 98CH36267)

She S, Lotufo R, Berger T, Wasowski A, Czarnecki K (2011) Reverse engineering feature models. In: ICSE

She S, Ryssel U, Andersen N, Wasowski A, Czarnecki K (2014) Efficient synthesis of feature models. Information and Software Technology 56(9). https://doi.org/10.1016/j.infsof.2014.01.012. http://www.sciencedirect.com/science/article/pii/S0950584914000238

Sinkala ZT, Blom M, Herold S (2018) A mapping study of software architecture recovery for software product lines. In: Proceedings of the 12th European Conference on Software Architecture: Companion Proceedings, ECSA '18

Slegers WJ (2009) Building automotive product lines around managed interfaces. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09. Carnegie Mellon University, Pittsburgh, pp 257–264

Software Engineering Institute (2008) Catalog of software product lines. http://www.sei.cmu.edu/productlines/casestudies/catalog/index.cfm

Stanciulescu S, Schulze S, Wasowski A (2015) Forked and integrated variants in an Open-Source firmware project. In: ICSME

Stanciulescu Ş, Berger T, Walkingshaw E, Wasowski A (2016) Concepts, operations and feasibility of a projection-based variation control systems. In: Proceedings of the 32nd International Conference on Software Maintenance and Evolution, ICSME'16

Staples M, Hill D (2004) Experiences adopting software product line development without a product line architecture. In: Proceedings of the 11th Asia-Pacific Software Engineering Conference, APSEC '04

Steger M, Tischer C, Boss B, Müller A, Pertler O, Stolz W, Ferber S (2004) Introducing pla at bosch gasoline systems: Experiences and practices. In: SPLC

Stoll P, Bass L, Golden E, John BE (2009) Supporting usability in product line architectures. In: Proceedings of the 13th International Software Product Line Conference, SPLC '09

Svahnberg M, Bosch J (1999) Evolution in software product lines: Two cases. J Softw Main 11(6):391–422

Takebe Y, Fukaya N, Chikahisa M, Hanawa T, Shirai O (2009) Experiences with software product line engineering in product development oriented organization. In: SPLC

Thiel S, Ferber S, Fischer T, Hein A, Schlick M, Bosch R (2001) A case study in applying a product line approach for car periphery supervision systems. In: SAE

Thörn C, Gustafsson T (2008) Uptake of modeling practices in SMEs: initial results from an industrial survey. In: MiSE

Thörn C (2010) Current state and potential of variability management practices in software-intensive SMEs: Results from a regional industrial survey. Inf Softw Technol 52(4):411–421

Thüm T, Apel S, Kästner C, Schaefer I, Saake G (2014) A classification and survey of analysis strategies for software product lines. ACM Comput Surv (CSUR) 47(1):6

Tischer C, Muller A, Mandl T, Krause R (2011) Experiences from a large scale software product line merger in the automotive domain. In: SPLC

Toft P, Coleman D, Ohta J (2000) A cooperative model for cross-divisional product development for a software product line. In: Proceedings of the First Conference on Software Product Lines : Experience and Research Directions: Experience and Research Directions. Kluwer Academic Publishers, Norwell, pp 111–132. http://dl.acm.org/citation.cfm?id=355461.355537

Vale T, de Almeida ES, Alves V, Kulesza U, Niu N, de Lima R (2017) Software product lines traceability: a systematic mapping study. Inf Softw Technol 84:1–18

Van Gurp J, Bosch J, Svahnberg M (2001) On the notion of variability in software product lines. In: Proceedings Working IEEE/IFIP Conference on Software Architecture

VEL (2018) Variability exchange language. https://www.variability-exchange-language.org/

Verlage M, Kiesgen T (2005) Five years of product line engineering in a small company. In: ICSE

Völter M, Visser E (2011) Product line engineering using domain-specific languages. In: de almeida, ES, Kishi, T, Schwanninger, C, John, I, Schmid, K (eds) Software Product Lines - 15th International Conference, SPLC 2011. IEEE Computer Society, Munich, pp 70–79. https://doi.org/10.1109/SPLC.2011.25

von Rhein A, Thüm T, Schaefer I, Liebig J, Apel S (2016) Variability encoding: From compile-time to load-time variability. J Log Algebr Methods Programm 85(1):125–145

Yin RK (2003) Case study research: Design and Methods, Applied Social Research Methods, vol 5, 3rd edn. Sage, London and Singapore

Zhang W, Jarzabek S (2005) Reuse without compromising performance: Industrial experience from rpg software product line for mobile devices. In: Proceedings of the 9th International Conference on Software Product Lines, SPLC'05. Springer, Berlin, pp 57–69. https://doi.org/10.1007/11554844_7

Zhou S, Stãnciulescu Ş, Leßenich O, Xiong Y, Wasowski A, Kästner C (2018) Identifying features in forks. In: Proceedings of the 40th International Conference on Software Engineering (ICSE)

Publisher's note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Thorsten Berger is an Associate Professor at Chalmers University of Technology and University of Gothenburg. His research focuses on model-driven software engineering, program analysis, and empirical software engineering, to develop methods and tools for highly configurable software. He received his Ph.D. from the University of Leipzig in 2013, then worked as a Postdoctoral Fellow at the University of Waterloo in Canada and the IT University of Copenhagen in Denmark. He participated in national and international research projects and received grants from competitive funding agencies (e.g., Swedish Research Council). He published in major software-engineering conferences (e.g., ICSE, FSE, ASE, OOPSLA) and journals (e.g., IEEE TSE, IEEE Software), and received best-paper awards at the 2015 ACM SIGPLAN conference on MODULARITY and the 2013 European Conference on Software Maintenance and Reengineering (CSMR, now IEEE SANER). He serves in the program committees of major conferences, including ICSE, FSE, and ASE, recently recognized with a distinguished reviewer award at ASE 2018.

Jan-Philipp Steghöfer is an Associate Professor at Chalmers University of Technology and the University of Gothenburg. He received his Ph.D. in 2014 from the University of Augsburg. His main research interests are software traceability, agile development of safety-critical systems, software product lines, and software engineering education. Jan-Philipp participates in several European projects with a focus on software traceability and drives development of the open source traceability management tool Eclipse Capra.

Tewfik Ziadi is currently an Associate Professor at Sorbonne Université and a researcher at Laboratoire d'Informatique de Paris 6 (LIP6). He received his Ph.D. from the University of Rennes 1 in 2005 and his habilitation (HDR) in 2016 from UPMC. His main research area of interest is related to Software Product Lines with different contributions published at ASE, SPLC or IST journal. He is a co-developer of the BUT4Reuse platform for Bottom-Up technologies for Reuse. He is the scientific coordinator of an international project and the general co-chair of the Systems and Software Product Line Conference (SPLC 2019) and co-chair of the ACM SAC VSPLE Variability and Software Product Line Engineering Track, 2019. He was the publication chair of ICSR 2018 (International Conference on Software Reuse), and co-chair of the editions of the REverse Variability Engineering workshop (REVE 2013-2018).

Jacques Robin received his PhD. in Computer Science from Columbia University in 1995. Since then he has been a research consultant at Bellcore Laboratories in New Jersey USA, an associate professor at Universidade Federal de Pernambuco in Recife, Brazil, a research laboratory manager at Thales Research and Technology, Palaiseau, France and a research engineer at Sorbonne Université and now Université PanthéonSorbonne both in Paris, France. His research has touched upon a wide variety of topics in both artificial intelligence and software engineering including natural language generation and computational linguistics, web information retrieval, machine learning and data mining, logic and constraint programming, formal software verification, model-driven engineering, process engineering, software product lines and contextaware self-adaptive systems. He is currently participating to multiple European projects on topics as varied as round-trip product lines engineering, verifiable artificial intelligence and selfadaptive cybersecurity.

Jabier Martinez joined the Digital Trust Technologies (TRUSTECH) area of Tecnalia in 2018. His interests are mainly related to modelling, software reuse, variability management, software product lines, and non-functional properties such as safety, security and privacy. After several years of industrial experience, he received his PhD in 2016 from the Luxembourg University (SnT, Interdisciplinary centre for Security and Trust) and Sorbonne University (Lip6, Laboratory of Computer Sciences, Paris 6). He participated in several European research projects. He coorganizes the Reverse Variability Engineering series of workshops.

Affiliations

Thorsten Berger[1] · Jan-Philipp Steghöfer[1] · Tewfik Ziadi[2] · Jacques Robin[3] · Jabier Martinez[4]

Jan-Philipp Steghöfer
jan-philipp.steghofer@chalmers.se

Tewfik Ziadi
tewfik.ziadi@lip6.fr

Jacques Robin
jacques.robin@univ-paris1.fr

Jabier Martinez
jabier.martinez@tecnalia.com

1 Chalmers | University of Gothenburg, Gothenburg, Sweden

2 Sorbonne University, Paris, France

3 University of Paris 1 Pantheon-Sorbonne, Paris, France

4 Tecnalia, Bilbao, Spain