Software and Systems Modeling
https://doi.org/10.1007/s10270-022-01011-2

REGULAR PAPER

Empirical analysis of the tool support for software product lines

José Miguel Horcas[1] · Mónica Pinto[1] · Lidia Fuentes[1]

Received: 30 April 2020 / Revised: 23 December 2021 / Accepted: 1 April 2022

Abstract
For the last ten years, software product line (SPL) tool developers have been facing the implementation of different variability requirements and the support of SPL engineering activities demanded by emergent domains. Despite systematic literature reviews identifying the main characteristics of existing tools and the SPL activities they support, these reviews do not always help to understand if such tools provide what complex variability projects demand. This paper presents an empirical research in which we evaluate the degree of maturity of existing SPL tools focusing on their support of variability modeling characteristics and SPL engineering activities required by current application domains. We first identify the characteristics and activities that are essential for the development of SPLs by analyzing a selected sample of case studies chosen from application domains with high variability. Second, we conduct an exploratory study to analyze whether the existing tools support those characteristics and activities. We conclude that, with the current tool support, it is possible to develop a basic SPL approach. But we have also found out that these tools present several limitations when dealing with complex variability requirements demanded by emergent application domains, such as non-Boolean features or large configuration spaces. Additionally, we identify the necessity for an integrated approach with appropriate tool support to completely cover all the activities and phases of SPL engineering. To mitigate this problem, we propose different road map using the existing tools to partially or entirely support SPL engineering activities, from variability modeling to product derivation.

Keywords Empirical analysis · Case studies analysis · Software product lines · State of the art · Tool support · Tooling road map · Variability modeling

1 Introduction

An increasing number of software application domains are adopting *Software Product Line* (SPL) approaches to cope with the high variability they present [1]. Examples of these domains are robotics [2], cryptography [3], operating systems [4], or computer vision [5]. However, the field of SPL is quite broad and constantly changing [6], with a large number of solutions available for each activity of an SPL. Moreover, these proposals are usually not properly integrated in common development practices, processes, or tool support. Thus, despite the number of successful stories about the use of SPL engineering,[1] the variability and reuse management problem has not yet been solved, and both the academy and the industry continue to experiment with their own solutions and approaches [7].

The success of an SPL approach depends on good *tool support* as much as on complete and integrated *SPL engineering processes* [8]. Regarding the processes, most SPL approaches typically cover the *domain* and *application engineering* processes [9], which include activities such as variability modeling and artifact implementation (domain engineering) and requirements analysis and product derivation (application engineering). However, the large number of approaches and extensions that exist for each activity [10] are usually not properly integrated among them and within the existing tool support. For instance, it is common to find SPL approaches that support basic variability modeling concepts (e.g., mandatory and optional features or includes and

Communicated by Joanne Atlee.

✉ José Miguel Horcas
horcas@lcc.uma.es

Mónica Pinto
pinto@lcc.uma.es

Lidia Fuentes
lff@lcc.uma.es

[1] CAOSD Group, ITIS Software, Universidad de Málaga, Andalucía Tech, Spain

[1] http://splc.net/hall-of-fame/.

Published online: 08 June 2022

excludes constraints), but it is more difficult that they support extended variability modeling (e.g., numeric and clonable features or multi-feature modeling). The same could be said for variability analysis, domain implementation, or product derivation. Moreover, some important activities, such as the analysis of *non-functional properties* (NFPs) or *quality attributes* and the *evolution* of SPL's artifacts [11], are set aside from existing SPL approaches. When considered, these activities are usually integrated into the traditional SPL process by reusing existing mechanisms which were not specifically designed for that purpose, for instance using attributes of extended feature models to specify quality attributes [12] while there are more appropriate approaches to deal with quality attributes, such as the NFR Framework [13].

Besides, although tool support is of paramount importance for the SPL management process [8], most existing tools cover only specific phases of the SPL approach (e.g., variability modeling or artifacts implementation). Those few tools that support several phases (e.g., FeatureIDE, pure::variants) [14] demand the adoption of an implementation technique such as *feature-oriented programming* (FOP) [15], *aspect-oriented programming* (AOP) [16], or *annotations* [17]; depend on the development IDE (e.g., Eclipse); or present some important limitations [18]. For instance, these limitations make the use of classical SPL approaches to web engineering challenging (e.g., FOP or AOP), mainly because of the nature of web applications that require the simultaneous use of several languages (JavaScript, Python, Groovy…) in the same application [19].

Unfortunately, few studies aim to understand the tool support across the different engineering activities of an SPL [20,21], and those that specifically focus on studying the tool support [8,22,23] usually report information extracted from the tool documentation or reference papers without really installing and using them with existing case studies. We have done this work with the overarching goal of empirically testing the tool availability, usability, and applicability. Our objective is to check out the existence of mature tool support for carrying out an SPL engineering process, especially in those application domains with complex requirements regarding SPL activities and variability modeling characteristics. For each activity in the domain and application engineering phases, we identify the requirements that tools should fulfill and analyze each tool's possibilities and limitations.

The paper answers the following Research Questions (RQs):

RQ1: *Which advanced variability modeling characteristics and SPL activities can be identified by analyzing case studies in the SPL community?* We answer this question by performing a *sampling study* where we select a sample of case studies in application domains with high variability, frequently used in the SPL community for research and evaluation. We extract the requirements of those case studies regarding variability and SPL activities, mainly focusing on advanced variability characteristics (Sect. 3).

RQ2: *What tools exist that provide support for the different phases of an SPL?* To answer this question, this paper presents an *exploratory study* of the SPL tools, focusing on their availability and usability and analyzing those tools that could be used to successfully apply an SPL approach (Sect. 4).

RQ3: *How do existing tools support the SPL engineering activities and variability modeling characteristics identified in RQ1?* We answer this question by *empirically analyzing* a subset of the tools identified in RQ2. We have selected it using availability and usability criteria. Then we analyze it, specifically focusing on those SPL activities and variability modeling characteristics that were previously identified during the analysis of the domains and case studies of the SPL community (Sect. 5).

RQ4: *Is it possible to carry out an SPL process, which includes the SPL activities and characteristics identified in the case studies analyzed, with the existing tool support?* That is, is it possible to cover all activities of complex approaches, including automatic reasoning, sampling of configurations, and evolution, among others? We answer to this question by defining different *roadmap* of tools that partially or completely support all phases of an SPL process (Sect. 6).

By answering these questions, the contribution of this paper is twofold. Firstly, SPL application developers and researchers will better understand up to what level the existing tools support is aligned with their application domains' requirements. Secondly, researchers can improve existing SPL processes, activities, and tools, so that they will be able to better plan their research in order to close the gaps that exist in the development of SPLs.

An earlier version of this work is published as a conference paper [24]. The former paper focuses on analyzing the tool support for a specific case study: WeaFQAs [25], studying whether WeaFQAs' variability characteristics could be modeled and managed with the current tools. In this article, we broaden the scope of our study to review a representative sample group of case studies' requirements. In particular, we have added an analysis and discussion of the variability characteristics and SPL activities required by up to 20 case studies in 6 different domains. Therefore, we have also updated our tool analysis to those requirements, including a new tool (i.e., analyzing 7 tools in total), and propose new road map for different levels of variability modeling expres-

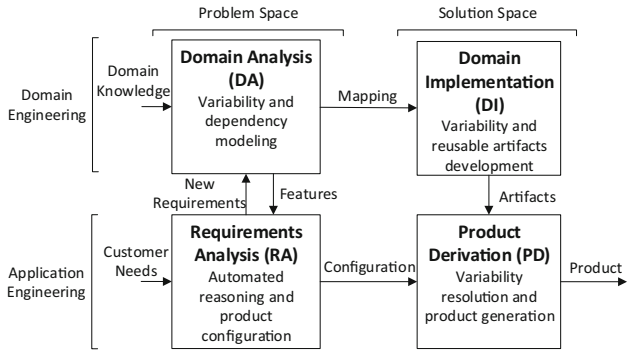

Fig. 1 The classical SPL approach with its processes and activities, adapted from Horcas et al. [24]

siveness and demanding SPL activities, such as sampling and optimization of configurations, among others.

The paper is structured as follows. Section 2 presents background information on SPL activities and variability modeling characteristics. Section 3 answers RQ1 by motivating our study, showing the requirements of complex domains and case studies. Section 4 answers RQ2 by presenting the state of the art of the existing tools for SPLs. Section 5 answers RQ3 by empirically analyzing a subset of those tools. Section 6 answers RQ4 by defining different tool road map to carry out all activities of an SPL approach. Section 7 discusses the threats to validity. Section 8 discusses related work, and Sect. 9 concludes the paper.

2 Background

This section presents the main processes and activities of an SPL approach and describes the different extensions and characteristics that have emerged over the years for each SPL activity.

The classical SPL approach [26] distinguishes between the *domain engineering* and the *application engineering* processes, with their main phases and activities (see Fig. 1): (1) variability and dependency modeling in the *domain analysis (DA)* phase; (2) automated reasoning and product configuration in the *requirements analysis (RA)* phase; (3) variability and reusable artifacts development in the *domain implementation (DI)* phase; and (4) variability resolution and product generation in the *product derivation (PD)* phase [9].

The following subsections provide more details about the activities in the different phases presented in Fig. 1. We put emphasis on the substantial number of extensions that have emerged throughout the years by referencing the most relevant articles or works where they were first proposed (see Fig. 2). Note that there are many more extensions, formalizations, languages, and concepts for SPLs and variability modeling. Here we briefly present those that are considered the most relevant and well accepted by the SPL community [10,27]. These concepts are used throughout the paper, firstly in Sect. 3, to identify the domain applications that require them, and then in Sect. 5, to analyze whether these concepts are covered or not by the existing tools.

2.1 Domain analysis (DA)

In the domain analysis phase, *feature models* (FMs) have been widely used to model variability since their introduction in FODA by Kang et al. [28]. From this work, different proposals have emerged for model variability and similar concepts (see top left of Fig. 2), such as orthogonal variability models (OVM) [26], probabilistic feature models [29], goal-based models [30], or decision models [31]. Even, there was an attempt at standardization with the definition of the *common variability language* (CVL) [32] and its extension, the *base variability resolution* (BVR) model [33], but it did not jell satisfactorily.

Due to the success of the FMs for variability modeling, a vast number of modeling languages and extensions have been proposed [10,34]. These are classified by some authors as basic variability modeling, extended variability modeling, and extra variability modeling.[2]

- **Basic variability modeling.** FODA [28] introduced the basic characteristics for modeling variability in FMs, such *basic features* as mandatory and optional features, alternative ("xor") and "or" groups, and *basic constraints* or relationships between features (e.g., requires and excludes constraints).
- **Extended variability modeling.** Well-known extensions of FMs include *variable features* or *non-Boolean values* such as *numerical features* [35,36] to represent numbers; *features with attributes* (called *extended-FMs*) [12] that provide more information about features, such as a cost attribute; *clonable features* or *multi-features* (called *cardinality-based FMs*) [37] that determine the number of instances of a feature that can be part of a product; and advanced relationships between features, such as *complex constraints* [38], which involve numerical features and multi-features.
- **Extra variability modeling.** Additional modeling mechanisms have been proposed to deal with more complex variability types. For instance, *feature viewpoints* [39] and *multi-perspective* [40] help to define multiple dimensions of variability separately (e.g., functionality, deployment, and context) [41]. Also, the combination of multiple product lines (called *MultiPLs*) [42] allows defining several families of products that are related among

[2] We follow the classification of variability modeling introduced by Alférez et al. [5].

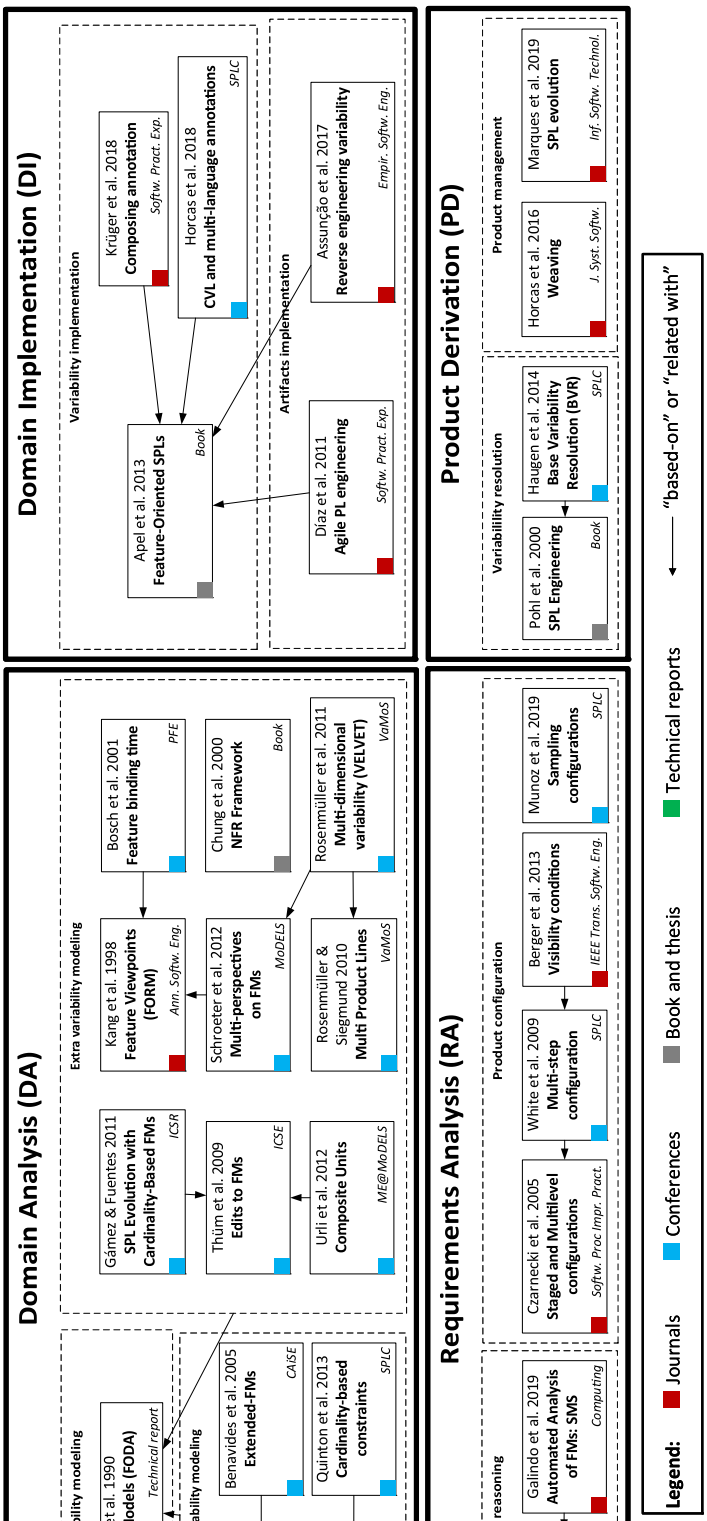

Fig. 2 Main concepts and extensions of variability modeling and SPL activities

them. Other extensions have been explicitly defined to deal with the modularization of large models and provide scalable models such as *hierarchical levels* and *compositions units* [43]; to deal with the evolution of models [44] using *refined FMs* or *edits to FMs* [45]; to handle *non-functional properties* (NFPs) such as the NFR Framework [13]; or to differentiate static and dynamic variability by defining *binding modes* such as binding states, units, or time [46].

2.2 Requirements analysis (RA)

The requirements analysis phase is in charge of analyzing the variability expressed in the FMs and creating a valid configuration by selecting the features that will form a specific product. Due to the complexity of dealing with large space configurations, some extensions have been proposed for automatic reasoning and product configuration (bottom left of Fig. 2).

- **Automatic reasoning.** Basic analysis on FMs includes model *statistics* and *metrics* such as *number of features*, *number of constraints*, and *type of features*. More complex analysis such as *model validation*, *model counting* (number of configurations), *anomalies detection*, and *explanations* require specific formalizations of the FMs [47]. Benavides et al. [48] enumerate more than 20 operators for automatic reasoning on FMs.
- **Product configuration.** Product configuration includes *feature selection*, *constraints propagation*, and *generation of configurations* either by *enumerating all the products* or by *sampling configurations* [36]. *Optimization of configurations* can also be achieved in this phase. Some extensions deal with the configuration process to make it more interactive and help the user to build a configuration product. Examples of these extensions are *staged and multilevel configurations* [37] to configure multiple dimensions or viewpoints; *multi-step and partial configurations* [49] that allow automatically deriving features and assist the user in the selection of features; and *visibility conditions* [4] that help to hide branches of the configurator hierarchy.

2.3 Domain implementation (DI)

In the domain implementation phase, developers build the reusable and variable artifacts of the SPL. There are several approaches and methodologies when it comes to implementing the artifacts and their variability (top right of Fig. 2).

- **Variability implementation.** There are different approaches to implement the variability of the reusable artifacts of an SPL [9]. Mainly, they can be divided in *composition-based approaches* and *annotation-based approaches* or a *combination of both approaches* [19,50]. Composition-based approaches include *component and service composition*, *design patterns*, *feature-orientation*, *aspect-orientation*, etc., while annotation-based approaches include *configuration parameters*, *preprocessors*, and *virtual separation of concerns*, among others [9].
- **Artifacts development.** The reusable (common or variable) artifacts of the SPL can be managed at different *abstraction levels*, from high abstraction models (software architectures, design diagrams…) to low level implementation details (code, functions, source files…). Extensions to the development of the SPL artifacts include different methodologies, such as *agile methods* [51] or *reverse engineering methods* [52]. Moreover, artifacts can be defined in *multiple languages* which can be used even in the same product [19].

2.4 Product derivation (PD)

The product derivation phase is in charge of generating or deriving the final product by resolving the variability specified in the product configuration. Additional activities have been proposed to manage the life cycle of the product after its generation (bottom right of Fig. 2).

- **Variability resolution.** This includes the *derivation of the product*, by resolving the variability of each variation point in the artifacts of the SPL according to the selected configuration of the feature model [33], and the *evaluation of the product* to check if it fulfills its requirements.
- **Product management.** Apart from resolving the variability and generating the final product, some extensions include the composition of different final products or *weaving* [53], the *traceability* of the features from the FMs to the artifacts in the final product, and the *evolution* of the SPL artifacts [54] and the *automatic propagation of changes* to the already configured products [55].

3 SPL and variability requirements

Variability modeling has been successfully applied in many domains, such as the automotive domain, computer vision, and software systems [56]. Analyses of how variability is managed in these domains, both conceptually and with respect to formalism and tool support, are important to understand the different challenges the domains pose and the level of support that existing proposals provide to deal with them. To identify these challenges and to motivate the rest of the paper, in this section, we answer our first research question:

RQ1: *Which advanced variability modeling characteristics and SPL activities can be identified by analyzing case studies in the SPL community?*
Rationale: There exist software systems that make intensive use of variability management techniques and can be customized for different scenarios [47]. Basic characteristics such as those introduced in FODA (Boolean features, optional and mandatory features, alternative and "or" groups, requires/excludes constraints) are not enough to model the variability of those systems. Thus, we need additional advanced variability mechanisms (e.g., numerical features, attributed features, multi-features, optimization of non-functional properties…). Our sampling study tries to find if there is a fundamental need to use advanced mechanisms to manage variability and identify those variability characteristics and activities.

To answer this question, we have selected a representative sample group from the studies mainly used in the SPL community, for research and evaluation. We have analyzed them by looking for variability requirements and uses of SPL concepts and variability mechanisms, in particular those introduced in Sect. 2.

Research method. We have conducted an empirical study consisting of a *sampling study* [57] in which we have selected a representative small group of case studies to analyze (a sample). In contrast to a systematic literature review where the state of the art is thoroughly reviewed, the sampling study aims for the representativeness of the selected case studies, which allows us to evidence the need to support the non-basic variability characteristics in current domains. To perform the sampling study, we define the following essential specific attributes according to the ACM SigSoft Empirical Standards [57]:

- **Goal of the sampling.** The main purpose of the sampling is to establish whether there is a real necessity of using advanced mechanisms to manage variability. Therefore, we are especially interested in those case studies that pose the most challenging requirements regarding variability; that is, case studies that make intensive use of variability management techniques beyond FODA concepts, requiring advanced variability mechanisms such as those introduced in Sect. 2.
- **Sampling strategy.** The sampling strategy consists of making an incremental selection of studies until we gather a representative sample of case studies evidencing the need to use advanced variability mechanisms. To identify the case studies, we manually searched the proceedings of the main research and industry tracks of the SPLC[3] and VaMoS[4] conferences, which are among the most relevant SPL and variability events, and then we used a snowball approach to the selected case studies. The search was limited from 2010 to December 2021 and performed in reverse chronological order to consider only the most updated versions of possible recurrent case studies. However, we also found older studies we considered during the snowball process. We found 477 articles, of which we selected a sample of 2-5 case studies per domain, limiting the sample to 6 domains and a maximum of 20 case studies. For the selection of the case studies, we used the following inclusion and exclusion criteria (IC and EC):

IC1: The paper presents a case study with requirements that involve the variability activities and variability concepts presented in Sect. 2.
IC2: The case study is described with a high level of detail about the variability characteristics it models and about the SPL activities it achieves or needs.
EC1: The case study requires only basic variability modeling (i.e., it only uses FODA concepts).
EC2: The case study requirements are a subset of another case study in the same domain.

We reviewed the articles in random order but guided by the domains. That is, we first randomly selected an article, identified its domain, and checked whether it meets our IC/EC. If the article did not pass the IC/EC, we randomly chose another one. If it passed the IC/EC, we focused on the requirements in the domain to which it belongs, looking for other articles in the pool with case studies of that domain. To do that, we relied on the title of the articles, on a snowball approach based on the references of the reviewed article that are already in the pool, and on our own experience (see biased judgment in Sect. 7). We stopped the incremental process when we reached a set of 2-5 case studies per domain, with a limit of 6 domains and a maximum of 20 case studies satisfying our IC/EC. This means that, from the starting pool of 477 papers, there probably were more than 20 studies fulfilling our IC that could be considered. However, we did not have to consider all of them, because we only needed a representative subset for our sampling goal. Note that the final objective is not to analyze the specific requirements of case studies or domains but to identify a need of using advanced variability modeling characteristics. Other samples from the same pool of papers that meet our IC/EC would also support our evidence. In contrast to a systematic literature or mapping study, we did not track the studies we left out due to the EC, because they are not relevant to the sampling

[3] https://dblp.uni-trier.de/db/conf/splc/.
[4] https://dblp.uni-trier.de/db/conf/vamos/.

study. Therefore, we did not need to collect information about the whole population or track the different filtering steps. We used Google Forms[5] to collect information about the case studies: name, primary reference, domain, year, type (industry, academic…), a brief description, and a list of variability and SPL requirements or challenges raised by the case study. These data were extracted from the information found in the primary reference paper that first introduced the case study or analyzed the case study from an SPL point of view.

- **Why the sampling strategy is reasonable?** Our hypothesis was that some case studies require advanced variability characteristics beyond the FODA concepts, and we needed to support it with a formal study. Finding just a few case studies of different application domains requiring advanced variability characteristics was enough to show the necessity of modeling or using those advanced mechanisms (our research question). However, to firmly support our hypothesis, we decided to identify between 2 and 5 case studies for each domain. As stated in Ralph et al. [57], the sampling strategy, despite not being necessary optimal, provides us with standard empirical research to identify those studies and answer our research question.
- **Rationale behind the selection of study objects.** In the sample, we included those case studies from research articles with requirements that aligned to those variability activities and variability concepts presented in Sect. 2. We did not differentiate between industrial and academic systems, since there are domains whose case studies pose significant challenges regarding variability, even if they are not considered in the industry yet. We show a preference for emergent domains (e.g., cyber-physical systems, computer vision) because we thought they would present more challenging variability requirements. But, in fact, evidence was easy to find in these domains. We realized that, in addition to these emergent domains, other domains that have been studied for years (e.g., operating systems) also pose challenging requirements regarding variability. We set 2010 as the starting date for the sampling study because most of the advanced variability concepts and characteristics used by the SPL community were defined or began to be used around 2010 or later (see Sect. 2). Thus, case studies requiring such characteristics started to appear on that date. Then, during the snowball process, we found older studies that we finally considered, in domains such as robotics.
- **Sample size.** We set the sample size to 20 case studies and 6 domains, selecting between 2 and 5 case studies per domain.

[5] https://forms.gle/PaN1L83jeW9yW7tM8.

The main artifacts developed that allow replicating and/or improving this analysis of case studies are available online.[6]

Results. The sample of 20 case studies from 6 different domains was analyzed in detail.[7] The case studies were grouped by application domains, and the results are presented in Tables 1-8. Firstly, Table 1 lists the analyzed domains and case studies in the order in which they were selected and analyzed, providing their reference and type (i.e., academic, industry…). During the analysis, we have searched for all the requirements listed in Table 2, which are organized according to the four main processes of an SPL (see Fig. 1) and the activities they include (see Fig. 2). This table summarizes the requirements and characteristics needed by each domain and has been generated as the union of the requirements of all the case studies in that domain. For a more detailed description of the case studies and their requirements, Tables 3 to 8 can be consulted. The rest of this section presents a brief discussion about the results, organized by domains. For each domain, we highlight the most relevant requirements regarding variability and SPLs and complement the information with the appropriate table that details all the requirements extracted for the analyzed case studies in that domain. We would like to highlight that the purpose of this study is not to draw conclusions about the characteristics of the domains themselves but instead to demonstrate that the advance variability requirements listed in Table 2 are present in a variety of existing and emergent domains.

Automotive domain (Table 3). The automotive industry has been associated for years with vehicles product lines. Nowadays, the complexity of such product lines has raised due to the heavy incorporation of intelligent software in autonomous vehicles. Here we describe some of the most relevant requirements of this domain. For instance, vehicles usually include electronic, mechanical, and software components, requiring *different viewpoints* with *complex constraints* involving technical and *architectural dependencies* [58]. These constraints are also introduced by commercial offers and stakeholder requirements, which give rise to the need of *MultiPLs* to distinguish two types of products (prototypes and commercial vehicles), which are different in terms of novelty, purpose, and the amount of reused assets. Moreover, case studies in this domain expose the needs of working at the *architectural level* and modeling *non-functional properties* such as the car efficiency or the safety traffic [59,60]. The complete set of requirements of the case studies in this domain are detailed in Table 3 and summarized in the first column of Table 2.

Computer vision domain (Table 4). Most of the case studies in this domain are related with the generation of syn-

[6] https://github.com/jmhorcas/SPLE-EmpiricalAnalysis.

[7] A .csv file with the information extracted for each case study is available in https://github.com/jmhorcas/SPLE-EmpiricalAnalysis.

Table 1 Domains and case studies analyzed

Domain	Case study	References	Type	Year
Automotive	Parking brake system	[58]	Industrial	2013
	Abstract fuel control system	[59]	Industrial	2019
	Autonomous vehicles	[60]	Academic	2018
Computer vision	Video generator	[5]	Industrial	2019
	Quality-based video generator	[61]	Industrial	2019
	Medical imaging workflow	[62]	Academic	2013
	Video surveillance system	[63]	Academic	2011
Cryptography	OpenCCE SPL	[64]	Industrial	2015
	Cryptography API	[3]	Academic	2019
	E-payment application	[65]	Academic	2016
Operating systems	Kconfig and eCos systems	[4]	Industrial	2013
	Android and Debian	[66]	Industrial	2014
Cyber-physical systems	Development approach to CPSs	[67]	Academic	2017
	Tank and quadcopter PLs	[68]	Industrial	2015
	Train control system	[69]	Industrial	2018
Robotics	Service robots	[2]	Academic	2019
	Autonomous mobile robots	[70]	Industrial	2012
	Cloud robotics SPL	[71]	Industrial	2014
	Landmark search	[72]	Industrial	2016
	Home service robots	[73]	Industrial	2005

thetic videos [5,61]. They show that, in the video domain, basic variability modeling (e.g., Boolean features) is not enough. They also demonstrate that modeling the variability in the video domain requires extended mechanisms such as *numeric features*, *multi-features* or *cardinality-based features*, and *complex constraints*. There are challenging requirements not only at the variability modeling phase but also in other phases, such as the *generation of optimal configurations* and the *reduction of the configuration space* to cope with models with large number of variants, as shown by all the case studies presented in Table 4. In fact, computer vision is one of the domains with the largest set of requirements for variability modeling and analysis, exposing the need of all the characteristics presented in Table 2 (column 2) for the domain and analysis phases.

Cryptography domain (Table 5). Cryptography is an algorithm-heavy domain used in thousands of software systems to protect any sensitive data they collect. There are different kinds of cryptography components (e.g., ciphers, digests, etc.), each suitable for a specific purpose and with various algorithms and configurations. Finding the right combination of algorithms and correct settings to use is often difficult [3]. Cryptography is also required by almost all electronic-based systems, such as e-payment systems and e-voting applications [65,74]. The encryption components need to be specifically customized to the application's requirements (e.g., the RSA algorithm with keys of 2048 bits) and then introduced (*weaving*) in the *software architecture* of the applications in a non-intrusive way (e.g., using an *aspect-oriented approach*). In Table 5, we can observe that this domain clearly requires advance variability management mechanisms such as the use of extended variability languages with *numerical features*, the *optimization of multiple objectives* during product configuration, the necessity of better *organizing large models* or the *weaving* of cryptography components with the application software architecture, etc. Table 2 (column 3) summarizes these requirements for the cryptography domain.

Operating systems domain (Table 6). Operating systems is one of the important domains where variability has been clearly identified and modeled [4]. Interestingly, the analyzed case studies reveal that the languages and models used in open-source operating systems (e.g., the Kconfig systems such as the Linux kernel and the Component Definition Language [CDL] used in the eCos system) use concepts that are beyond core FODA concepts. These range from the use of *domain-specific vocabulary* (e.g., *tristate* features) [75] to *binding modes* for *static and dynamic variability*. They also have in common the need of dealing with *larger models* and *high numbers of dependencies* between features. Table 6 details all the requirements of the case studies in this domain, while column 5 in Table 2 summarizes them.

Cyber-physical systems domain (Table 7). Cyber-physical systems (CPSs) describe autonomous and adaptable systems such as embedded systems, which integrate sensors and actuators to monitor, control, and influence

Table 2 Domain requirements

	Requirements and Characteristics	Automotive	Computer vision	Cryptography	Operating systems	Cyber-physical systems	Robotics
Domain analysis (DA)	**Basic variability modeling**						
	Basic features (optional, mandatory, or, xor)	■	■	■	■	■	■
	Basic constraints (requires, excludes)	■	■	■	■	■	■
	Extended variability modeling						
	Variable features or non-Boolean values (numerical, enumerations, string, date…)	□	■	■	■	■	□
	Extended FMs (features with attributes)	■	■	□	■	■	■
	Default values, deltas, ranges, and precision	□	■	■	■	■	□
	Cardinality-based FMs (clonable features or multi-features)	□	■	□	□	■	■
	Complex constraints (involving numerical features, attributes, multi-features…)	□	■	■	■	■	□
	Extra variability modeling						
	Multi-dimensional variability and Multi Product Lines (feature viewpoints, multi-perspectives…)	■	■	□	□	■	■
	Modularization of larger models (composition units, hierarchical levels…)	□	■	■	■	■	□
	Evolution of FMs (refined FMs, edits to FMs…)	□	■	■	□	□	■
	Non-functional properties (goals, operationalizations, contributions…)	□	■	□	□	■	■
	Binding modes (time: static, dynamic; state…)	□	■	□	■	■	■
	Metainformation (documentation, descriptions, annotations…)	■	■	□	■	■	□
	Other extensions (arbitrary group multiplicity, constraints between viewpoints, abstract features…)	■	■	□	□	■	■
Requirements analysis (RA)	**Automatic reasoning**						
	Basic analysis of FMs (statistics, metrics…)	■	■	■	■	■	■
	Analysis operations on FMs (validation, model counting, anomalies detection…)	■	■	■	□	□	□
	Product configuration						
	Generation of configurations (features selection, enumeration, constraints propagation…)	■	■	■	■	■	■
	Sampling configurations	□	■	□	□	■	□
	Optimization of configurations	■	■	■	□	■	□
	Interactive configuration process (multi-step and partial configurations, derived features, visibility conditions…)	■	■	■	■	□	□

Table 2 continued

	Requirements and Characteristics	Automotive	Computer vision	Cryptography	Operating systems	Cyber-physical systems	Robotics
Domain implementation (DI)	**Variability implementation**						
	Composition-based approach (components, feature-oriented, aspect-oriented…)	■	■	■	■	■	■
	Annotation-based approach (parameters, stereotypes, virtual separation of concerns…)	■	□	□	■	■	□
	Combined approach (composition and annotations)	□	□	□	■	■	□
	Artifacts development						
	High abstraction level (architecture, design, models…)	■	□	■	■	■	■
	Low abstraction level (code, functions, files…)	□	□	■	■	■	□
	Multi language artifacts (language independence)	□	□	□	■	□	■
Product derivation (PD)	**Variability resolution**						
	Product derivation	■	■	■	■	■	■
	Product evaluation	■	□	■	□	■	□
	Product management						
	Weaving or composition of products	□	□	■	□	□	□
	Traceability of features	■	□	□	□	□	□
	Evolution changes (automatic propagation of changes)	■	□	■	□	□	■

■ The characteristic is present in most of Table 1 case studies in the domain.
□ The characteristic is not present in almost any of Table 1 case studies in the domain.

Table 3 Requirements of the automotive domain

Automotive studies
Case study: Parking brake system [58]. **Type:** Industrial. **Year:** 2013
Description: An Electric Parking Brake (EPB) system commonly used by automotive companies (e.g., Renault) to replace or improve the functionality of conventional parking brake systems
Requirements:
* *Documentary language* to represent and describe vehicle features for each vehicle family
**Different viewpoints and constraints between them* to cover the scope of the system architecture (e.g., electronic, mechanical, and software components)
* A modeling language to provide representations for each of the viewpoints (e.g., SysML/UML with custom profiles)
* *MultiPLs* to distinguish two types of products, which are different in terms of novelty, purpose, and the amount of reused assets: prototypes and commercial vehicles
* Updating domain models to reflect changes (i.e., *support for evolution*)
* *Staged configuration* process that follows the different viewpoints
* Ensuring *traceability* of the variability source in respect to where it appeared in the system.
Case study: Abstract fuel control system [59]. **Type:** Industrial. **Year:** 2019
Description: Model that takes as input signals the pedal angle and engine speed and outputs the signal air-fuel ratio (AF), which influences fuel efficiency and car performance
Requirements:
* *MultiPLs* to model cars, families of cars, and even families of families of cars, which share common features, hazards, environmental conditions, and driving conditions
* *Multiple viewpoints with relationships among them* such as hazards, cars features, system control, and context (e.g., environment)
* *Numerical features* such as the pedal angle, engine speed, sensors' tolerance, or atmospheric pressure with different *range in intervals* (e.g., $[0, 61.2)$ for the pedal angle) and *precision*
* *Constraints* among parameters (e.g., the best sensors with high precision are only available on sport cars)
* *Non-functional properties* such as fuel efficiency and car performance
* *Structural links* from the feature models to the underlying Simulink models
* *Data variability* represented as parameters in the models, and *composite variability* as model variant blocks or variant subsystems
* *Devising and reasoning* about different testing/simulation conditions
Case study: Autonomous vehicles [60]. **Type:** Academic. **Year:** 2018
Description: A reconfigurable vehicle controller agent for autonomous vehicles that adapts the parameters of a car-following model at runtime, in order to maintain a high degree of traffic quality (efficiency and safety) under different weather conditions
Requirements:
* Several *numerical features* (with decimal point), such as speed, acceleration, jam distance, maximum safe deceleration, and time headway
* *Viewpoints and constraints between them* involving the environmental context (e.g., type of road, weather conditions, traffic density), the autonomous vehicle (e.g., behavioral parameters and car-following models), and the traffic quality attributes (e.g., efficiency and security)
* *Non-functional properties* such as efficiency (e.g., travel time, group disagreement) and safety (e.g., time to collision, lane change rate)
* *Structural links* from the feature models to the underlying software architecture.

Table 4 Requirements of the computer vision domain

Computer vision case studies
Case study: Video generator [5]. **Type:** Industrial. **Year:** 2019
Description: Generation of variants of synthetic videos used to benchmark video processing algorithms
Requirements:
* *Numeric features* such as the number of vehicles that appear in a video
* *Attributes* with *default values*, *deltas* to discretize continuous domain values, and *multi-ranges*
* *Multi-features* or *cardinality-based features* to configure, for instance, different vehicles in different ways for the same video
* *Complex constraints* (e.g., a countryside scene implies including less than 10 vehicles)
* *Metainformation*
* *Differentiating between static and dynamic variability*
* *Sampling and filtering relevant configurations* to control the automated reasoning about them
* Generation of *optimal configurations* regarding different criteria and using different optimization techniques
Case study: Quality-based video generator [61]. **Type:** Industrial. **Year:** 2017
Description: Generation of synthetic videos used to benchmark computer vision systems
Requirements:
* *Numeric features*, *ranges for attribute domains*, and *complex constraints*
* Reduce the configuration space to avoid the generation of non-acceptable videos (e.g., too noisy or too dark)
* *Sampling of configurations* and machine learning are needed to create predictive models or classifiers to infer the properties of still non-generated configurations
* *Variability of different dimensions and the constraints among them* to generate context-aware configurations
Case study: Medical imaging workflow [62]. **Type:** Academic. **Year:** 2013
Description: Definition of a domain-specific language
Requirements:
* Coping with *large-scale feature models*
* *Multi-dimensional or multi-perspective approach* (e.g., hardware, software)
* Various *levels of abstraction*
* Composition and management support for a set of *interrelated models*
* *Multi-step configuration* support
Case study: Video surveillance system [63]. **Type:** Academic. **Year:** 2011
Description: Use of the domain-specific language defined in [62]
Requirements:
* Coping with *large feature models*
* *Multi-perspective approach*
* Variability at design and at runtime

Table 5 Requirements of the cryptography domain

Cryptography case studies
Case study: OpenCCE SPL [64]. **Type:** Industrial. **Year:** 2015
Description: Variability of the Java Cryptography Architecture (JCA) and development of the OpenCCE SPL to manage cryptography components
Requirements:
* *Hierarchical structure* to differentiate symmetric and asymmetric ciphers and algorithm classes (e.g., encryption ciphers vs. hashing digests)
* *Non-Boolean values* to specify attributes such as key size or memory consumption levels
* *Reusing through some form of referencing* to avoid redefining the same attributes for each cipher
* *Automated reasoning* to *find instances* and *optimize multiple objectives*
* *Guide developers* through selecting the relevant cryptography components to use
* *Automatic generation of the required code* with the correct API calls for them
* *Analyzes the final program* to ensure that no threats have been introduced, either during *initial development* or during *program evolution*
Case study: Cryptography API [3]. **Type:** Academic. **Year:** 2016
Description: Extension of the work in [64] to model generic cryptography components
Requirements:
* *Multiple product types* such as encryption tasks and encryption algorithms that need to be modeled separately but with *constraints between the products* (e.g., algorithms of a task)
* *Numerical variables* such as the size of the produced hash or the encryption key size in bits
* *Limits* (e.g., the output size), *allowed ranges* (e.g., key sizes values between 512 and 65,536 for RSA), and *default values* (e.g., to provide the user with average secure defaults)
* *Ordinal attributes* such as security or performance levels of an algorithm defined as *enumeration types* (e.g., "*Slow*," "*Strong*," etc.) instead of integers encoding *discrete values* (e.g., 1 to 4)
* *Partial configurations* to specify certain properties and leave the instance generator to decide based on a set of constraints
* *Optimizing numerical features* for properties such as key size or iterations of the cipher
* *Ignore irrelevant parts of the model* based on the kind of task that is being configured
Case study: E-payment application SPL [65]. **Type:** Academic. **Year:** 2016
Description: Customization of encryption components according to the application requirements and weaving with the software architecture of the application.
Requirements:
* *Customization* of encryption components according to the application's requirements (e.g., use of the RSA algorithm with keys of 2048 bits)
* *Injection or weaving* of the encryption components in the application *software architecture* in a non-intrusive way (e.g., using *aspect-orientation*)

Table 6 Requirements of the operating systems domain

Operating systems case studies
Case study: Kconfig and eCos systems [4]. **Type:** Industrial. **Year:** 2013
Description: Study of variability modeling languages used in open-source operating systems. Concretely, Kconfig systems such as the Linux kernel, and the Component Definition Language (CDL) used in the eCos system
Requirements:
* *Domain-specific vocabulary* (e.g., *tristate* features in Kconfig systems)
* *Scaling variability modeling* since the *models are significantly larger* (e.g., 1,708 features on average), and have a very *different shape*
* *Data features*, including *integers, floats, and strings* (e.g., strings are used for file names)
* *Constraints*, including *arithmetic and string operators.*
* *Derived (computed) features* to simplify constraints and to perform calculations that otherwise would be hidden in the build system
* *Group constraints*, being mostly *xor* (1..1) and a few *mutex* (0..1)
* Three-valued logic *binding modes* to specify whether a feature implementation is linked statically, built for dynamic linking, or absent
* *Default values*, either represented as literals or expressions to save the user unnecessary configuration work
* *Visibility conditions* to hide whole branches of the configurator hierarchy
Case study: Android and Debian [66]. **Type:** Industrial. **Year:** 2014
* **Description:** Extension of work in [4] to include Android, Debian, and Eclipse
Requirements:
* *Static variability mechanisms* such as *code generators* and *conditional compilation* to selectively compile sources files
**Dynamic variability mechanisms* such as *configuration parameters* and *component-oriented architectures* to load components according to a specific configuration
**Binding modes*, and in particular, *binding times* to allow basic units or composite units to be installed and removed at runtime
**Modularization mechanisms* to deal with *larger models* and *high number of dependencies*

Table 7 Requirements of the cyber-physical systems domain

Cyber-physical systems studies
Case study: Development approach to CPSs [67]. **Type:** Academic. **Year:** 2017
Description: A vision on a development approach to variable cyber-physical systems
Requirements:
* *Multiple viewpoints* for different variability aspects: components (hardware, behavior), context (environment, use case), hierarchy (intramodel, intermodel), quality (performance, safety), and time (design, runtime)
**Complex constraints* to model interactions between the viewpoints
* *Heterogeneity to the variability modeling*: source-code fragments, hardware components, or documentation
* *Dynamic SPLs* for self-adapting and reconfiguration of components due to changes in their environment
* *Optimization of configurations* to find the most suitable configuration in particular use cases and environments
* *Non-functional properties* to achieved defined goals based on qualitative properties
Case study: Tank and quadcopter PLs [68]. **Type:** Industrial. **Year:** 2015
Description: An SPL to control the liquid level (water or chemical products) of a configurable tank system and an SPL of unmanned aerial vehicles (UAVs) to configure a quadcopter
Requirements:
* *Multiple viewpoints* and *hierarchical levels* to represent different variability taxonomies: interaction (sensors and actuators), mechanical elements, hardware, and software
* *Complex constraints* between the variation points of the different viewpoints (e.g., a mandatory sensor to measure the acidity if the tank's liquid is chemical)
* *Non-functional properties* to model the quality of the sensors and actuators (sensitivity, ranges, response time), and the performance and energy consumption of the global system
**Numerical features* to model parametric values of sensors that are manually set up by the users
* *Cardinality-based features* to instantiate the same piece of software once or more, for example, to control several physical processes or to obtain data from different sensors
* Work at the *architectural level* considering resource constraints when implementing variability in the software architecture
* *Sampling of configurations* to verify and validate sample configuration systems, involving techniques to choose configurations where different features interact among them at least once (e.g., t-way testing)
Case study: Train control system [69]. **Type:** Industrial. **Year:** 2018
Description: The European Railway Traffic Management System (ERTMS) is an international standard to improve the interoperability, performance, reliability, and safety of modern railways. ERTMS relies on the European Train Control System (ETCS), which is an automatic train protection (ATP) system that continuously supervises the train, ensuring that the safety speed and distance are not exceeded.
Requirements:
* *Hierarchical levels* to model the variability of the different levels of operation
* *Numerical features with accuracy and constraints between them and normal features.* For example, to specify the maximum distance that a train is allowed to travel or the maximum allowed speed depending on the track morphology
* Other *complex constraints* related to *dynamic variability* such as *temporal constraints* (e.g., temporary speed restrictions and conditional or unconditional emergency stops)

Table 8 Requirements of the robotics domain.

Robotics case studies
Case study: Service robots [2]. **Type:** Academic. **Year:** 2019
Description: Variability engineering in various academic and industrial service robots
Requirements:
* *Multitude of variability dimensions*, such as the robot mechanical structure, the tasks to be performed, and the environmental conditions (context)
* *Dependencies between the dimensions* (e.g., different navigation algorithms are used based on the kinematics of the robot)
* *Different variants of the same robot model* to deploy different software based on the customization that is required by a specific customer
* *Multi SPLs* to design each functional subsystem of an autonomous robot as an SPL
* *Modeling, composing, and deriving the software architecture* of each sub-systems in terms of model-driven development
* A substantial degree of variability can only be resolved *at runtime* (e.g., obstacles of various kinds may require some flexibility in the behavior of the robot)
* *Different operation modes* (e.g., a sharing mode attribute to allow the end user to decide how the functionalities of the robot should be shared).
Case study: Autonomous mobile robots [70]. **Type:** Industrial. **Year:** 2012
Description: Variability of autonomous mobile robots at the architecture level using domain-specific languages and focusing on non-functional properties
Requirements:
* *domain-specific languages* (e.g., SysML and xADL) to represent core assets at the *architectural level*
* *Non-functional properties* represented as *attributes* in the architecture: stimulus, source of stimulus, environment, artifact, response, response measure, and expected response
Case study: Cloud robotics SPL [71]. **Type:** Industrial. **Year:** 2014
Description: Definition of a cloud robotics SPL
Requirements:
* *Multiple viewpoints* are required to model different dimensions of the robot
Case study: Landmark search robot [72]. **Type:** Industrial. **Year:** 2016
Description: Definition of landmark search robots with variability
Requirements:
* Variability is managed at the *architectural level*
Case study: Home service robots [73]. **Type:** Industrial. **Year:** 2005
Description: Definition of home service robots with variability
Requirements:
* *Refined feature models* to add features extracted from legacy home service robots by applying a reverse engineering process.

physical objects [67]. Due to the variety of technologies involved in the development of the CPSs' devices, they require very diverse variability characteristics and SPL activities such as *multiple viewpoints* and *hierarchical levels* for different aspects (e.g., context, sensors, actuators, software, etc.); *dynamic variability* with *complex constraints* for self-adaptation and reconfiguration; *cardinality-based features* to instantiate multiple sensors; *optimization* of *non-functional properties* such as energy consumption, among other requirements detailed in Table 7. This variety of requirements makes CPSs one of the most complex domain to deal with from the point of view of SPLs (see column 4 in Table 2).

Robotics domain (Table 8). Robotics systems are a specific type of CPS. Although they share some of the requirements of CPS, robotic technologies are characterized by high variability, where each robotic system is equipped with a specific mix of functionalities [2]. This is another domain in which advanced variability management mechanisms are required. It is important to highlight some of them, such as the use of *MultiPLs* for each subsystem of an autonomous robot [2], the *architectural-level derivation* of products [72], or the explicit representation of *non-functional requirements* as part of the variability modeling [70]. In Table 8, we can observe all the requirements in detail for the analyzed case studies, while Table 2 (last column) summarizes them for this domain.

We will finish this section summarizing the answer to RQ1:

Conclusions and lessons learned from RQ1: There is an important number of highly relevant domains in which advanced variability characteristics beyond FODA [28] were identified, and complex SPL activities (e.g., sampling, optimization) [76,77] are required by existing case studies. In particular, numerical features, attributes, and complex constraints involving numerical values are required by almost all domains, while the activities related to the analysis of configurations (e.g., multi-step configuration, optimization) are often demanded by current domains. Another common requirement has been managing the systems from a high abstraction level by modeling the variability at the architectural level.
The analysis of the requirements exposed in the sample of case studies shows the need to consider the variability and requirements listed in Table 2 when building an SPL approach. Thus, the general conclusion is that, independently of the characteristics of each specific domain, there is an important number of existing and emergent domains in which advanced variability characteristics and SPL activities are demanded.

4 State of the art of SPL tools

Providing tool support for all the requirements extracted in the previous section (Table 2) is challenging for SPL researchers and developers. Our first step is to explore the existing tool support for SPL by answering our second research question:

RQ2: *What tools exist that provide support for the different phases of an SPL?*
Rationale: To analyze whether the SPL tools provide support for advanced variability mechanisms or not, we first need to identify the existing tools providing some support to SPLs. *This exploratory study will identify the existing tools providing some support to SPLs, classifying them according to the SPL phases they cover.*

We analyze the current state of the art of SPL tools to identify which ones are available online and are really usable for researchers and the SPL community. The goal is to collect all possible tools related to SPL to check their status (available, working, updated, usable) before considering them for analysis. This does not pretend to be a systematic review of tools but an exploratory study to identify existing tools.

Research method. We performed an exploratory study, which Ralph et al. [57] define as "an empirical inquiry that investigates a contemporary phenomenon (the 'case') in depth and within its real-world context". The cases in our approach are tools, and our goal is to perform an in-depth study of these tools' characteristics, in the real-world context of case studies that demand a series of advanced variability characteristics. Our exploratory study consists of a manual search on different sources. First, we identified SLRs [8,21,22] and surveys [23] about SPL tools. We also searched the proceedings of the Demonstrations and Tools track in some of the most relevant events about SPL and variability (e.g., SPLC,[8] VaMoS[9]) for the period not covered by the SLRs and surveys (2015-2019). The only inclusion criteria (IC) we applied was the following:

IC1: The tool is directly related to SPL or is used in the context of SPL to provide support to at least one of the phases of the SPL process: DA, RA, DI, and PD, as defined in Sect. 2 and in Apel et al. [9] and Pohl et al. [26].

Any other tool not considered for downloading and testing was directly discarded without registering in the data

[8] https://dblp.uni-trier.de/db/conf/splc/.

[9] https://dblp.uni-trier.de/db/conf/vamos/.

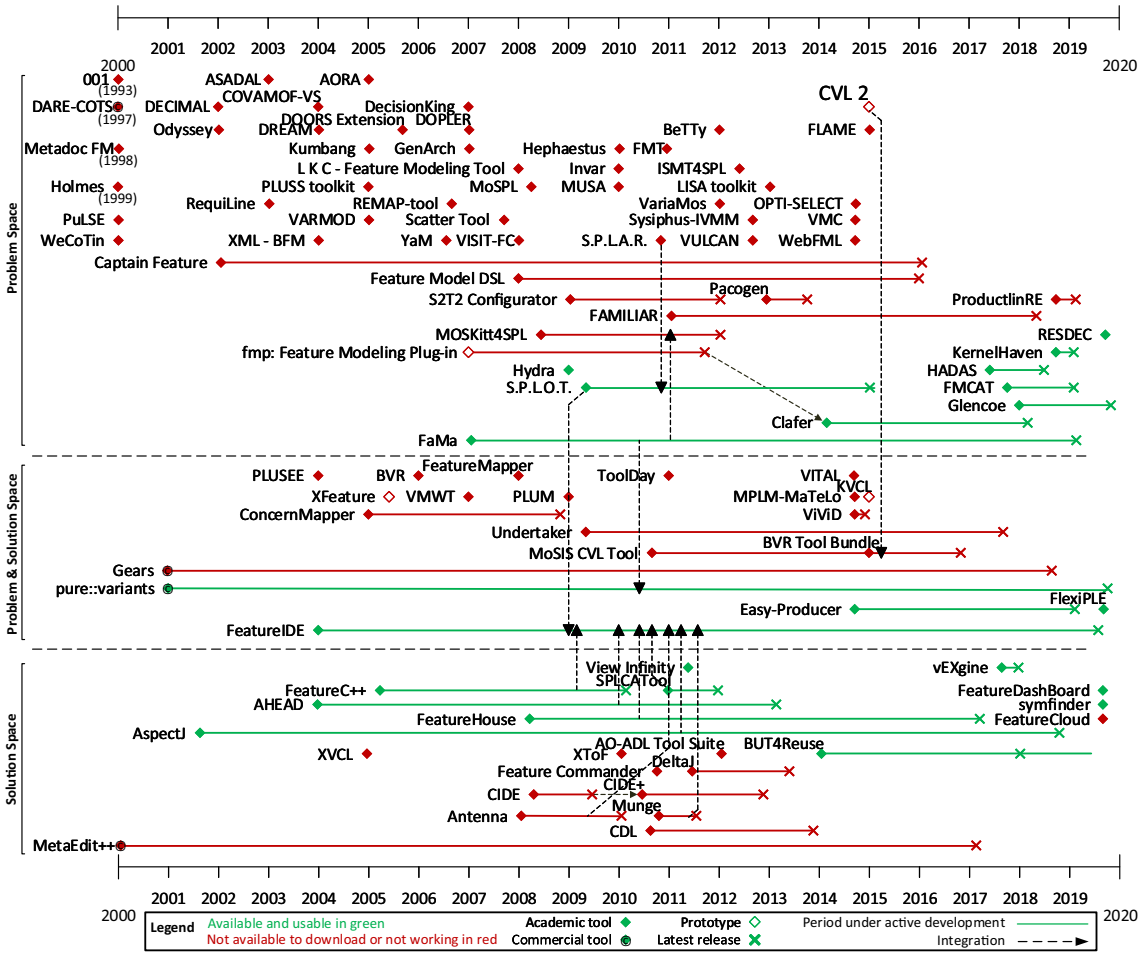

Fig. 3 State of the art of SPL tools

extraction form. For each reported tool, we searched for its availability (i.e., its website, code repository, or executable). When the information was not available in the paper, we performed a manual search on web search engines (e.g., Google) to localize the tool by applying the following search strings: «`name of the tool`», `tool`, `SPL`, `Software Product Line`, and `variability`. Finally, we downloaded, installed, and launched each tool to check its correct functioning and main use case.

Data extraction form. We used Google Forms[10] to capture the basic information about the availability of the tools: name, brief description, URL, main reference, SPL's phases covered, type of tool (academic, commercial, prototype), first and last release date, availability, current status, and integration with other tools. These data have been extracted from the information found in the reference papers, the official websites, and the code repository of the tool. The main artifacts developed that allow replicating and/or improving this state of the art are available online.[11]

Results To illustrate the state of the art, we have built a timeline (Fig. 3) with all the SPL tools published until December 2019.[12] As summarized in Fig. 4 and at the top of Table 9, only 6% of them cover all phases of the SPL process (Problem & Solution Space block in the middle of Fig. 3). Moreover, there seems to be more interest in the problem space than in the solution space since the DA (72% of the tools) and the RA (64%) are the phases most covered by the tools (top of Fig. 3). The DI and PD phases are only covered by 38% and 14% of the tools, respectively (bottom of Fig. 3). These values can be explained due to the difficulty of building tools that support all the functionalities required by an SPL approach across all the SPL activities, particularly

[10] https://forms.gle/JfH9bKHHTgCLc31R7.

[11] https://github.com/jmhorcas/SPLE-EmpiricalAnalysis.

[12] A .csv file with the tools information is available in https://github.com/jmhorcas/SPLE-EmpiricalAnalysis. The original timeline published on Horcas et al. [24] contained 97 tools. In this work, we have updated the timeline including tools suggested by other researchers and increased the number of tools considered up to 103 tools.

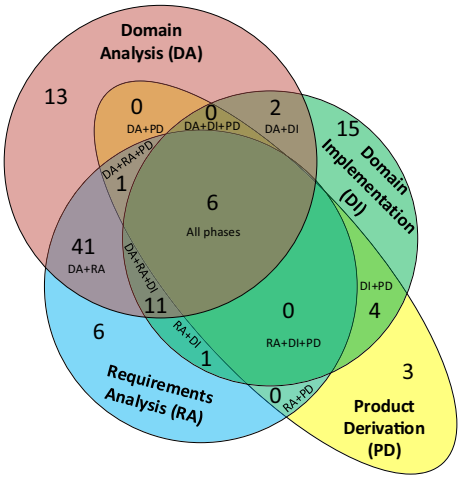

Fig. 4 SPL phases covered by existing tools

those related to SPL activities dealing with large configuration spaces or the generation and derivation of products, which are considered well-known NP-problems [9].

We also found evidence that there are a large number of tools that are academic (91%). The reason behind this is that practitioners often propose new tools when they are making research on the SPL field, and thus the percentage of academic vs. industrial tools is so disproportionate. However, many of the academic tools are usually abandoned shortly after the associated research project ends. The tool becomes usually obsolete, is no longer available to be downloaded, or becomes non-usable due to the continuous evolution of their core technologies (e.g., Java). This fact can be observed in the multiple red points on the top of the timeline in Fig. 3.

We conclude this section with our answer to RQ2:

Conclusions and lessons learned from RQ2: There are many tools (we discovered up to 103) that provide some support for SPL, most of them academic. However, researchers are often not aware of all these tools and the kind of support they provide to SPL activities and therefore continue proposing new tools to support their contributions in SPL and abandoning them later, especially when the contribution of the tool is too specific and has not been integrated as part of another tool (e.g., FeatureIDE).
Our study gives a comprehensive vision of the current state of the art of the SPL tools and helps users to be aware of the existing tools and the SPL phases each tool supports. Therefore, the user can select appropriate tools according to their needs.

5 Tools support analysis for complex SPLs

This section answers our third research question, selecting a subset of tools identified in Sect. 4:

RQ3: *How do existing tools support the SPL engineering activities and variability modeling characteristics identified in RQ1?* **Rationale:** The lack of mature tool support is one of the main reasons that make the industry reluctant to adopt SPL approaches. The problem becomes worse when considering advanced variability mechanisms such as those identified in Sect. 3 for several case studies since practitioners are not aware of which tools will provide support for those characteristics or how the tools support them. By answering this research question, we aim to help SPL users to choose the tool that offers the best support according to the variability characteristics they need to model and the activities they need to carry out within an SPL. *Our exploratory study will analyze what kind of support the existing tools provide for the SPL activities and variability characteristics identified in RQ1.*

5.1 Tool selection

Of the 103 tools discovered when seeking an answer to RQ2 (Sect. 4), we included in the analysis all tools that meet the following inclusion criteria (IC):

IC1: The tool is fully available and usable, that is, it can be downloaded, installed, and successfully executed.

This inclusion criteria is met by 23 tools (22%) (see bottom of Table 9). Note that multiple academic tools did not pass our IC1. Many of them are abandoned soon after the associated research project ends. The tool becomes obsolete, stops being available to be downloaded, or becomes non-usable due to the technical debt [88]. In the case of industrial tools such as Gears or MetaEdit++, these tools are not freely available, since no evaluation or limited version is provided, in contrast to, for example, the pure::variants tool, which offers an evaluation version. Working with industrial tools requires contacting distributors for tool assistance, and sometimes no evaluation or academic versions are available. This lack of free evaluation versions usually prevents SPL researchers from knowing if the tool is appropriate for their needs before acquiring an expensive license. To select the tools to be finally analyzed in detail, we executed the 23 tools so as to identify their main functionalities and use cases regarding the SPL activities and characteristics identified in Sect. 3. Then, we apply the following exclusion criteria (EC) to those 23 tools:

Table 9 Summary of tools for SPL by phases covered

Tools	Phases covered											
	Problem space				Solution space							
	DA	%	RA	%	DI	%	PD	%	All	%	Total	%
Total	74	72	66	64	39	38	14	14	6	6	103	100
Academic	67	91	60	91	35	90	10	71	3	50	94	91
Commercial	3	4	2	3	2	5	2	14	1	17	4	4
Prototype	4	5	4	6	2	5	2	14	2	33	5	5
Available and usable	11	15	12	16	11	28	7	50	2	33	23	22
Academic	10	91	10	83	10	91	6	86	1	50	21	91
Commercial	1	9	1	8	1	10	1	14	1	50	1	5
Prototype	0	0	1	8	0	0	0	0	0	0	1	5

EC1: The tool is a prototype, a preliminary or beta version without a stable release.[13]

EC2: The tool has been integrated within another tool that has already been selected.

EC3: The tool supports only a specific activity or characteristic of an SPL phase (e.g., optimization of non-functional properties). That activity or characteristic is also covered by another selected tool also supporting other activities and characteristics.

EC4: The tool relies on another SPL tool to offer its functionality (e.g., performance analysis). The former is not a tool specifically designed to support the development of an SPL process.

We have chosen the seven SPL tools to be analyzed in this section by applying these exclusion criteria. Table 10 summarizes these tools, showing their main reference, the year of its first release, the date of its last update, the SPL phases covered by the tool, the website from where it can be downloaded (or accessed in case of an online tool like SPLOT or Glencoe), its code repository if available, and a brief description of the tool. Note that many other tools are available, such as FeatureHouse [89] or AHEAD [90], but EC2 has excluded them since they are integrated within other tools like FeatureIDE [14]. Others, such as Hydra [91] or ProductlineRE [92], did not pass IC1, since they do not have a stable release. Although they can be executed, they present several bugs during execution because of third-party dependencies or currently obsolete specific versions of plugins (i.e., technical debt), so they did not pass IC1. Others are exclusive to a particular domain, such as FMCAT [93] that focuses on the analysis of dynamic services product lines, and those activities are also supported by other tools such as FeatureIDE [14] or pure::variants [80], so they did not pass IC3. Finally, other tools such as HADAS [94] offer a specific functionality related to SPL (e.g., estimation of energy consumption of configurations) but rely on other SPL tools such as Clafer [81] which provides the core functionality regarding the SPL activities, so they did not pass IC4.

[13] A *stable release* (also called *production release*) is the last product version that has passed all verifications/tests, and whose remaining bugs are considered acceptable.

5.2 Experiments

To perform our empirical analysis of the selected tools, we have tried to model the variability characteristics identified in Sect. 3, adapting the modeling to the support provided by the different tools when the tool does not provide direct support to model or implement that characteristic. It is worth remembering that the objective of this analysis is not to model all the case studies identified but to analyze whether the tools provide support to model those characteristics. All artifacts developed and used throughout the different phases are available online to replicate the experiments.[14] These include: (1) the FMs in several formats: SPLOT, Clafer, GFM, v.control, pure::variants, Excel, SPASS, and DIMACS; (2) the software components implemented with different variability approaches: annotations with Antenna, feature-oriented programming with FeatureHouse, and aspect-oriented programming with AspectJ; (3) the software architecture models in UML; and (4) other artifacts such as model to model transformations that implement specific variation points.

The experiments were performed on two desktop computers: (1) Intel Core i7-4770, 3.40 GHz, 16GB of memory, Windows 10 64 bits and Java 8 and (2) Intel Core i7-4771, 3.5GHz, 8GB of memory, Windows 7 64 bits and Java 8.

5.3 Tool analysis

In this section, we analyze the selected tools to check whether they satisfy the requirements of the different domains identified in Sect. 3. For each phase in the SPL process, we explain

[14] https://github.com/jmhorcas/SPLE-EmpiricalAnalysis.

Table 10 Description of the selected SPL tools

S.P.L.O.T.

Reference: Mendonca et al. [78]. **Year:** 2009. **Last update:** Jan. 2015

SPL phases covered: DA, RA

Website: http://www.splot-research.org/

Description: Online tool to edit, debug, analyze, configure, share, and download FMs. It offers hundreds of FMs from academics and practitioners

Glencoe

Reference: Anna Schmitt [79]. **Year:** 2018. **Last update:** Nov. 2019

SPL phases covered: DA, RA

Website: https://glencoe.hochschule-trier.de/

Description: Web application to work with variability models. Model importation from DIMACS or `PV` [80]. Several solvers for the automated analysis of FMs

Clafer

Reference: Antkiewicz et al. [81]. **Year:** 2014. **Last update:** Feb. 2018

SPL phases covered: DA, RA

Website: https://www.clafer.org/ **Repository:** https://github.com/gsdlab/clafer

Description: General-purpose lightweight language for structural modeling: feature modeling and configuration, class and object modeling, and metamodeling. Clafer includes several solvers and model reasoners, such as the constraint programming library Chocosolver [82]

`FAMA`

Reference: Benavides et al. [83]. **Year:** 2007. **Last update:** Feb. 2019

SPL phases covered: DA, RA

Website: https://www.isa.us.es/fama/ **Repository:** http://famafw.github.io/FaMA/

Description: Framework for automated analyses of FMs integrating some of the most commonly used logic representations and solvers proposed in the literature (BDD, SAT, and CSP solvers)

FIDE

Reference: Leich et al. [84], Thüm et al. [14]. **Year:** 2004. **Last update:** Nov. 2021

SPL phases covered: DA, RA, DI, PD

Website: http://www.featureide.com/

Repository: https://featureide.github.io/

Description: Open-source Eclipse framework with plug-in-based extension mechanism to integrate and test existing tools and SPL approaches such as FeatureHouse, AHEAD, AspectJ, or Antenna, among others [85]

`PV`

Reference: Spinczyk and Beuche [80]. **Year:** 2001. **Last update:** Dec. 2021

SPL phases covered: DA, RA, DI, PD

Website: https://www.pure-systems.com/

Description: Commercial solution that supports all phases of the SPL process. Many extensions that connect pure::variants with common systems and software engineering tools [86]

`VEX`

Reference: Horcas et al. [87]. **Year:** 2017. **Last update:** Jan. 2018

SPL phases covered: DI, PD

Website: http://caosd.lcc.uma.es/vexgine/

Repository: https://github.com/jmhorcas/vEXgine

Description: Customizable implementation of the CVL execution engine [32] that can be extended with custom transformation engines to support multiple variability approaches

Table 11 Tool support for the requirements and characteristics of each activity of an SPL

	Requirements and Characteristics	S.P.L.O.T.	Glencoe	Clafer	FAMA	FeatureIDE	pure::variants	vEXgine
Domain Analysis (DA)	**Basic variability modeling**							
	Basic features (optional, mandatory, or, xor)	■	■	■	■	■	■	□
	Basic constraints (requires, excludes)	■	■	■	■	■	■	□
	Extended variability modeling							
	Variable features or non-Boolean values (numerical, enumerations, string, date…)	□	□	■	□	□	⊠	□
	Extended FMs (features with attributes)	□	□	⊠	■	⊠	■	□
	Default values, deltas, ranges, and precision	□	□	⊠	⊠	□	⊠	□
	Cardinality-based FMs (clonable features or multi-features)	□	□	⊠	□	⊠	⊠	□
	Complex constraints (involving numerical features, attributes, multi-features…)	□	□	■	⊠	□	■	□
	Extra variability modeling							
	Multi-dimensional variability and Multi Product Lines (feature viewpoints, multi-perspectives…)	□	□	⊠	□	⊠	⊠	□
	Modularization of larger models (composition units, hierarchical levels…)	□	□	⊠	□	⊠	■	□
	Evolution of FMs (refined FMs, edits to FMs…)	□	⊠	■	□	⊠	■	□
	Non-functional properties (goals, operationalizations, contributions…)	□	□	⊠	⊠	□	⊠	□
	Binding modes (time: static, dynamic; state…)	□	□	⊠	⊠	□	⊠	□
	Meta-information (documentation, descriptions, annotations…)	□	□	⊠	⊠	⊠	⊠	□
	Other extensions (arbitrary group multiplicity, constraints between viewpoints, abstract features…)	□	⊠	⊠	□	⊠	⊠	□
Requirements Analysis (RA)	**Automatic reasoning**							
	Basic analysis of FMs (statistics, metrics…)	■	■	⊠	■	■	■	□
	Analysis operations on FMs (validation, model counting, anomalies detection…)	■	■	⊠	■	■	■	□
	Product configuration							
	Generation of configurations (features selection, enumeration, constraints propagation…)	⊠	■	⊠	⊠	■	■	□
	Sampling configurations	□	□	■	□	■	□	□
	Optimization of configurations	□	□	⊠	□	⊠	□	□
	Interactive configuration process (multi-step and partial configurations, derived features, visibility conditions…)	■	⊠	⊠	□	■	⊠	□

Table 11 continued

	Requirements and Characteristics	S.P.L.O.T.	Glencoe	Clafer	FAMA	FeatureIDE	pure::variants	vEXgine
Domain Implementation (DI)	**Variability implementation**							
	Composition-based approach (components, feature-oriented, aspect-oriented…)	☐	☐	☐	☐	■	■	■
	Annotation-based approach (parameters, stereotypes, virtual separation of concerns…)	☐	☐	☐	☐	■	■	☒
	Combined approach (composition and annotations)	☐	☐	☐	☐	☒	☒	■
	Artifacts development							
	High abstraction level (architecture, design, models…)	☐	☐	☐	☐	☒	☒	■
	Low abstraction level (code, functions, files…)	☐	☐	☐	☐	■	■	☒
	Multi-language artifacts (language independence)	☐	☐	☐	☐	☒	■	■
Product Derivation (PD)	**Variability resolution**							
	Product derivation	☐	☐	☐	☐	■	■	■
	Product evaluation	☐	☐	☐	☐	■	■	☒
	Product management							
	Weaving or composition of products	☐	☐	☐	☐	☒	☐	■
	Traceability of features	☐	☐	☐	☐	■	■	☒
	Evolution changes (automatic propagation of changes)	☐	☐	☐	☐	☐	☒	☒

■ It completely supports the requirement/characteristic.
☒ It partially supports the requirement/characteristic.
☐ It does not support the requirement/characteristic.

how the tools provide practical support for the activities and characteristics in that phase and discuss our findings. Table 11 summarizes the results of our analysis.

5.3.1 Domain analysis (DA) phase

As described in Sect. 2, this phase is in charge of modeling the domain variability. Almost all tools (except **vEXgine**) provide support for model the variability using FMs. **vEXgine** is based on CVL [32], and despite the fact that its CVL metamodel supports several of the considered characteristics (e.g., variable features and clonable features), the tool **vEXgine** mainly focuses on the solution space phases (DI and PD)

Basic variability modeling. All tools supporting the domain analysis phase allow building basic FMs.

- **Basic features.** **Glencoe** and **FeatureIDE** offer an excellent graphical editor to build the diagram of the FM following the notation proposed by Czarnecki [95], while **S.P.L.O.T.**, **pure::variants**, and **FAMA** provide a great tree-based reflective editor. In **Clafer**, the FM needs to be created using a text editor. In all tools, mandatory, optional, and group features ("or" and "xor") are supported.
- **Basic constraints.** Each tool provides its own notation to define cross-tree constraints, but all of them support at least the requires and excludes constraints.

Extended variability modeling. The support for extended characteristics is very limited. While **S.P.L.O.T.** and **Glencoe** do not implement extended characteristics, other tools provide their own implementation, which often does not completely fit with the definition most widely accepted by the SPL community [37]. For instance, the support for variable features (or non-Boolean features) and the support for feature with attributes are confused in some tools because of the thin difference between these two concepts (variables and attributes).

- **Variable features or non-Boolean values.** Only **Clafer** provides full support for specifying variable features with a specific type (e.g., integer) that behaves as a normal feature but allows providing a value during the configuration step, for example, a numerical feature to represent the key size of an encryption algorithm. In **pure::variants**, variable features can be defined using features with attributes.
- **Extended FMs.** **FAMA** and **pure::variants** offer complete support for defining features with attributes, for example, to specify a utility value for each feature in the FMs. To support attributes in **Clafer**, we have to rely in the Clafer Multi-Objective Optimizer (ClaferMOO) [81], which is a specific reasoner for attributed-FMs, or in the modeling of the attributes as variable features. The latter implies defining an additional variable feature (e.g., integer) for each attribute associated with each normal feature and making sure those variable features are selected in the final configuration. **FeatureIDE** supports attributes only partially, because it requires selecting the "Extended Feature Modeling" composer, and then, no other composer can be selected. Also, using the extended models of **FeatureIDE**, only the variability modeling activity is supported since they are not compatible with the graphical editor or the later analysis, and attributes need to be manually defined in the XML source file.
- **Default values, deltas, ranges, and precision.** There is no explicit support for these characteristics in any of the analyzed languages, despite the fact that these characteristics are required by most of the case studies analyzed, as shown in Table 2. However, it is possible to provide default values to variable features or to attributes by defining constraints (see support for complex constraints). But this solution does not allow to change the value during configuration. Deltas, ranges, and precision can also be simulated by manually defining constraints or additional features (e.g., discretizing a variable) at the expense of losing information.
- **Cardinality-based FMs.** Clonable features or multi-features are the most difficult characteristics to be implemented, and thus, no tool provides support for them completely, although this is also a required characteristic in many domains as shown in Table 2. **Clafer** allows cloning any feature in the FMs and configuring each instance, but this is done at the configuration step and deciding whether a feature is clonable should be done at the domain analysis phase. **FeatureIDE** and **pure::variants** allow a similar behavior of clonable features by inserting subtrees in the FMs. In **FeatureIDE**, this characteristic follows the VELVET approach of MultiPLs [41], while **pure::variants** introduces the concept of "variant instance" as a link in the FMs to another configuration space. Within this approach, and in contrast to **Clafer**, the number of instances for the clonable feature has to be decided in the domain analysis phase and not at the configuration step, where this decision is normally taken.
- **Complex constraints.** Only **Clafer**, **FAMA**, and **pure::variants** allow specifying constraints about the values of non-Boolean features (numerics). Constraints in **pure::variants** are based on Prolog or a variant of OCL: pvSCL [80], so in **pure::variants** it is possible to specify constraints that are more complex. **Clafer** also allows specifying basic constraints (and, or, not, implies) over features that can be cloned later. Once again, the results shown in Table 11 demonstrate that the support currently provided by the analyzed

tools is not aligned with the domain requirements shown in Table 2.

Extra variability modeling. There is very poor support for extra characteristics of variability modeling.

- **Multi-dimensional variability and Multi Product Lines.** No tool provides explicit support for defining variability in different dimensions such as feature viewpoints or multi-perspectives. However, `pure::variants` and `Clafer` offer some mechanisms to modularize FMs that can be used to model separately the variability of different dimensions (see the following point about modularization of large models). On the other hand, supporting MultiPLs is more an organizational concept rather than an extra variability modeling facility. However, `FeatureIDE` provides explicit support for the development of the technical aspects of MultiPLs by following the VELVET approach [41], but this extension is still in its infancy.
- **Modularization of large models.** Large FMs cannot be easily modularized within existing tools by means of composition units or hierarchical levels. `Clafer` allows defining multiple FMs as abstract classes, but all of them must be in the same file. `FeatureIDE`, as discussed before for clonable features and multi-dimensional variability, supports MultiPLs that can help to modularize entire SPLs, but the FMs themselves cannot be divided in multiple files. In `pure::variants`, the support is better since it defines a "hierarchical variant composition" to link an FM inside another.
- **Evolution of FMs.** Modifications and edits to FMs once created can be complex in some tools like `S.P.L.O.T.`, `Glencoe`, and `FeatureIDE`, where modifying a part of the feature model usually can only be achieved by removing that part and adding it again. Contrarily, `pure::variants` and `Clafer` allow even moving features from a branch to another in a straightforward way.
- **Non-functional properties.** No tool provides explicit support for dealing with NFPs. That is, modeling goals, subgoals, operationalizations of goals, and contributions between them [13]. However, we can rely on features with attributes (in `pure::variants` and `FAMA`) and variable features (in `Clafer`) to model basic NFPs of the FMs, such as cost or performance, and define constraints between them.
- **Binding modes.** As occurs with NFPs, there is not explicit support for specifying binding modes, but it can also be simulated using attributes (`pure::variants` and `FAMA`) or variables (`Clafer`).
- **Metainformation.** There is also no explicit support for documenting the FMs by adding descriptions or annotations to the features or using domain-specific vocabulary. An alternative is the use of comments in the source file of the FM.
- **Other extensions.** Each tool provides additional characteristics for variability modeling. For instance, `Glencoe` and`pure::variants` allow mixing mandatory features within "or" groups. `Glencoe`, `pure::variants`, and `Clafer` support arbitrary multiplicity in group features (e.g., `x..y`, where `x` can be distinct from `1` and `y` distinct from `*`). `FeatureIDE` and `Clafer` allow defining abstract features. More complex constraints such as constraints between different viewpoints are not supported in any tool.

Discussion. `S.P.L.O.T.` and `Glencoe` are the most usable tools for the domain analysis phase since they are available online, intuitive, and easy to use and even their models can be exported to `FeatureIDE` and `pure::variants`, respectively. However, they do not provide any support for advanced characteristics. Only `Glencoe` and `FeatureIDE` use the notation proposed by Czarnecki [95], which is now the most comprehensible and flexible (and the most used) [47]. The notation of `Clafer` can be tedious for variability modeling, although it provides good support for variable features and acceptable support for clonable features. `S.P.L.O.T.` and `pure::variants` share a similar interface to build the FMs, following a tree structure but each of them with its own notation. It is worthy to mention that there are other tools that provide explicit support for clonable and variable features such as the tools that provide support to the CVL language [32], for example, the MoSIS CVL Tool [96] and the BVR Tool Bundle [97]. However, those tools are specific to the CVL language and are currently obsolete or not available.

Regarding some advanced characteristics, first, it is worthy to differentiate between variable features [36,98], which are those that require providing a value (e.g., integer, string, float) during configuration, and features with attributes [12]. A value change in an attribute does not suppose a different configuration of the FMs, because an attribute assigned to a feature is not a variation point of an artifact in the SPL. This distinction should be considered in the tools. Second, the cardinality of the clonable features or multi-features should be defined in the domain engineering phase of the SPL, while the specific number of instances for a clonable feature should be specified in the application engineering phase. Neither `Clafer` nor `pure::variants` follow these criteria. Third, there are more appropriate approaches for modeling NFPs than encoding them as attributes. For instance, the NFR

framework [13], which allows defining goal, sub-goals, operationalizations, and contributions of the NFPs and whose integration in an SPL tool can be desirable. Finally, regarding complex constraints, tools should provide support for defining high-order logic constraints in standard constraint languages such as OCL and programming languages such as Prolog (as in **pure::variants**). Those constraints should be able to be defined using any kind of feature (variable features, clonable features) or even between features defined in different FMs as for multi-dimensional variability.

5.3.2 Requirements analysis (RA) phase

The goal of this process is to select a desired combination of features according to the application requirements. This phase should also consider the automatic analysis of the variability model and managing configurations of the product at the feature level.

Automatic reasoning. Analysis of variability is one of the most important activities in an SPL, and thus, all tools covering the RA phase provide some kind of support for automatic reasoning of FMs.

- **Basic analysis of FMs.** Statistics and metrics about FMs are provided by almost all tools in different degrees. **Glencoe** is the best tool in this sense, showing up to 27 metrics about the FMs (e.g., core features, optional features, number of constraints, deep of the tree diagram, average children per feature, homogeneity of features, etc.). **FeatureIDE** and **pure::variants** also offer great statistics and even distinguish between the metrics of the FM and the metrics of the SPL implementation. In contrast, **Clafer** is the tool that provides less information with only 5 metrics.
- **Analysis operations on FMs.** **FAMA** is the tool that stands out here because it was built with the purpose of performing FM analyses. Thus, it supports most of the operations defined in Benavides et al. [48]. These operations cover model validation (consistency, void feature model…), anomalies detection (dead features, false-optional features, redundancy constraints…), and model counting (number of configurations), among others. Depending on the requested analysis, each tool uses a specific FM formalization and/or solver to perform the analysis. For example, to calculate the number of configurations or the variability degree of the feature model, **S.P.L.O.T.** uses a Binary Decision Diagram (BDD) engine [99] for which counting the number of valid configurations is straightforward [100]. **Glencoe** uses a Sentential Decision diagram (SDD) [101] engine that enables determining the total number of configurations within very short times. Within **pure::variants** is also possible to calculate the number of configurations for each subtree under a selected feature. The other tools (**Clafer** and **FeatureIDE**) require to generate all configurations in order to enumerate them, and thus, with these tools it is not possible to calculate the number of configurations for large models (e.g., 10^{30} configurations) in a reasonable time.

Product configuration. Managing configurations and products includes activities such as sampling and optimizing configurations, as well as assisting application engineers when generating configurations. However, tools fail to provide good support for these activities and generally focus on a basic generation of a configuration by selecting features from the variability model.

- **Generation of configurations.** Although all tools allow generating a configuration by selecting a set of features, only **S.P.L.O.T.** and **Glencoe** provide automatic derivation of features due to the cross-tree constraints. Regarding the enumeration of configurations, generating all configurations of a large model is infeasible for any tool nowadays. For instance, using the Choco solver [82] integrated in **Clafer**, it takes 1 hour to generate $13e6$ configurations from a total of $5.72e24$ (calculated with **S.P.L.O.T.**), requiring more than a billion years to generate all configurations. **FeatureIDE**, in addition, can generate the associated code of all the products (for small FMs).
- **Sampling configurations.** Only **Clafer** and **FeatureIDE** allow to sample a specific number of configurations, but the process is not completely random [36].
- **Optimization of configurations.** None of the selected tools provide specific good support for finding optimal configurations (based on NFPs) in FMs. **Clafer**, with its ClaferMOO module, provides a multi-objective optimization mode, but this implies to use another kind of model not related to the **Clafer**'s variability model. **FeatureIDE**, with the use of plugins, allows a complete configuration based on the optimization of NFPs and historical data [102].
- **Interactive configuration process.** The support to manage partial configurations and step-by-step configurations varies a lot among tools. **S.P.L.O.T.** provides validation and statistics of partial configurations and also automatic derivation of features and auto-completion of the configuration with fewer features or the configuration with more features. This is done through an online step-by-step configuration assistant. **Glencoe** allows generating partial configurations by assisting the user with colors over the feature diagram as the user selects the desired features, including the automatic derivation of features due to the cross-tree constraints. **Clafer** allows

generating complete configurations from a partial one thanks to its instantiation process based on constraint definition. **FeatureIDE** and **pure::variants** allow generating partial configurations and calculate the number of valid configurations from those partial configurations. **FeatureIDE** also integrates a visual guide for product configuration [103] that assists the user with colors over the feature diagram and recommendations as the user selects the desired features with auto-completion of the configuration, including the automatic derivation of features due to the cross-tree constraints.

Discussion. Although existing tools provide good support for the RA phase, there are some activities that are not properly covered. Firstly, none of the tools allows generating all configurations efficiently for large variability models (10^{30} configurations) like the ones used in some domains (e.g., operating systems). Secondly, existing tools are able to calculate the number of configurations but without taking into account advanced characteristics like variable features or clonable features, which considerably increments the total number of configurations. Finally, the support provided by the analyzed tools to generate optimal configurations of products based on some criteria like NFPs [104] is not straightforward. Thus, it is necessary to use additional plugins, such as the ClaferMOO module for **Clafer**, or external tools such as SPLConqueror [105] in combination with **FeatureIDE** for the analysis of colossal feature models considering sampling and optimization of configurations (e.g., analysis of NFPs). The extension mechanism of **FeatureIDE**, based on plugins, and the provided API allow applying specific optimization techniques (e.g., evolutionary algorithms), additional formalizations of the FMs such as CNF [106], or the use of advanced SAT solvers [107,108]. Most of these applications have been developed as part of a research work and are available as evaluation or proof of concept artifacts. They still require to be properly integrated in a main release of a tool like **FeatureIDE** to make them widely available to the SPL community.

5.3.3 Domain implementation (DI) phase

This phase focuses on the implementation of the SPL variable and common artifacts (e.g., models and code). Only **FeatureIDE**, **pure::variants** and **vEXgine** cover this phase. While **FeatureIDE** and **pure::variants** are tools based on the Eclipse IDE and provide all the necessary support to implement an SPL (i.e., project and file manager, integrated editors, etc.), **vEXgine** offers a standalone application to resolve the variability in the models provided by its interface, but this is the unique tool that provides support for CVL.

Variability implementation. Several variability implementation techniques have been widely studied in the SPL [9], and most of them have been successfully incorporated into the analyzed tools.

- **Composition-based approach.** **FeatureIDE** provides good support for different variability approaches. Concretely, it supports feature-oriented programming using the FeatureHouse approach or AHEAD and aspect-oriented programming with AspectJ, among others [109]. Furthermore, **FeatureIDE** offers a plugin-based mechanism to incorporate any other approach into the IDE. **pure::variants** provides its own variation points system which is also compatible with multiple approaches such as Aspect-Oriented Programming (e.g., AspectJ and AspectC++). **vEXgine** provides a complete set of variation points with associated model transformations to resolve the variability following an orthogonal approach [87]. For example, it defines an "object existence" variation point to determine the existence or absence of an artifact in the SPL.
- **Annotation-based approach** **FeatureIDE** provides support for specific annotation-based approaches like Antenna (Java comments), Colligens (C preprocessor), or Munge (Android). **pure::variants** in contrast, support annotations for different generic languages (e.g., Java, JavaScript, C++). **vEXgine** does not provide support for annotations by default, because it is a composition-based tool, but annotations can be supported by extending it in a combined approach (see below).
- **Combined approach** Neither in **FeatureIDE** nor in **pure::variants**, it is possible to combine different approaches in different parts of the application (e.g., annotations and AHEAD). Actually, only the combination of FeatureHouse with Java and AspectJ is supported in **FeatureIDE**. In **vEXgine**, it is possible to use and combine different variability mechanisms (composition and annotations) [19], but the resolution of that variability must be delegated to an external engine [87].

Artifacts implementation. Artifacts can be implemented at different abstraction levels, from elements in software architectures and design models to pieces of code, functions, or resource files. Moreover, a product usually is composed by artifacts defined in different languages. In general, the tools analyzed provide good support for defining and/or managing the product's artifacts.

- **High abstraction level.** **pure::variants** and **vEXgine** offer the best support for working at the architectural and design levels. However, **pure:: variants** requires the commercial version to manage high abstract models (e.g., UML), while

vEXgine requires to define the appropriate model transformations, although it supports any Meta-Object Facility (MOF)-compliant model [87]. FeatureIDE offers the possibility of combining FeatureHouse and UML, but actually, this integration is not completely operable.
- **Low abstraction level.** FeatureIDE and pure::variants work by default at the code level providing good support for implementing the SPL artifacts (as discussed for the composition and annotation-based approaches). In contrast, vEXgine needs specific extensions to work at the code level [19].
- **Multi-language artifacts.** vEXgine is completely independent of the language used to implement the artifacts at the architectural or code level. FeatureIDE and pure::variants support multiple programming languages, but it is not easy to combine them in the same project.

Discussion. FeatureIDE and pure::variants are excellent tools to build the artifacts of an SPL from scratch, but it is very difficult to apply the variability mechanisms (e.g., AOP, FOP) to existing third party libraries. pure::variants also allows extracting variability from source code, but most of the advanced options of pure::variants are only available in the commercial version [86]. Moreover, no tool supports an effective variability mechanism to be applied over several languages (Java, Python, JavaScript) in the same project.

5.3.4 Product derivation (PD) phase

Variability resolution and product derivation are achieved only with the tools analyzed that cover this phase (i.e., FeatureIDE, pure::variants, and vEXgine), although these tools present some limitations in the management of products after their generation.

Variability resolution. This includes generating the final product (by resolving the variability of the artifacts according to the selection of features made in the RA phase) and validating the generated product.

- **Product derivation.** All the three tools (FeatureIDE, pure::variants, and vEXgine) can resolve the variability specified in FMs over SPL artifacts to generate a final product.
- **Product evaluation.** The code resulting from the products generated with FeatureIDE and pure::variants can be directly compiled and validated. In contrast, in vEXgine, the user needs to manually verify if the generated models are valid and conform to the associated metamodel.

Product management. When a final product is generated, it can be incorporated within another product (e.g., in the case of subsystems) by applying some combination mechanism (weaving or MultiPL). In addition, the traceability of features and the propagation of changes in the final products when the requirements change or domain artifacts evolve need to be considered.

- **Weaving or composition of products.** Only vEXgine provides complete support for weaving products by defining custom model transformations [25]. The flexibility of pure::variants allows integrating other tools like Git to partially support mixing variants [110]. FeatureIDE does not support explicit weaving of final products but integrates the VELVET approach [41] for MultiPL, which may be used to weave the products, although this is a prototype and in this case the product derivation is not fully operable.
- **Traceability of features.** FeatureIDE provides several mechanisms that facilitate tracing features such as feature colors, naming, or virtual separation of concerns [111]. In pure::variants, its *family model* [80] allows describing the variable architecture/code and connecting it to the FM via appropriate rules. vEXgine allows connecting the features of the FMs directly with the artifacts of the SPL through explicit references to the variation points, but the final product does not contain this information.
- **Evolution changes.** The support for propagating changes in the variability model to the existing configurations exists but is limited. FeatureIDE does not provide explicit support for evolution, and the products need to be generated again after changes in the SPL artifacts. In pure::variants, the source code of the product, variants can be evolved by using merge operations from Git [110,112]. vEXgine can evolve the deployed artifacts with the help of specific model transformations and evolution algorithms [55,113], but the effort of defining those transformations is considerable.

Discussion. Variability resolution and product derivation are achieved by all the analyzed tools. A limitation in FeatureIDE is that only one composer (e.g., FeatureHouse, annotations) can be selected for an SPL application, and thus, the combination of different approaches requires building and integrating a custom composer within FeatureIDE. Apart from resolving the variability and generating the final products, existing tools have not paid special attention to advanced activities such as weaving or evolution. However, those activities could be incorporated into some tools thanks to their extension mechanisms, such as the possibility of defining new composers in

FeatureIDE [85] or the custom engines and model transformations of **vEXgine** [87].

Conclusions and lessons learned from RQ3: While most of the tools analyzed provide full support for basic variability modeling, they present several limitations when dealing with more complex variability requirements. For example, SPLOT, Glencoe, and FeatureIDE are recommended tools for modeling variability, but they do not support advanced variability characteristics. In contrast, Clafer and pure::variants, which provide support for advanced variability characteristics, implement such support differently. Practitioners should be aware of what kind of support they need for their projects. In addition, the support for advanced SPL activities such as sampling configurations or optimization of configurations is scarce due to the difficulty of managing and dealing with large feature models and configuration spaces. Here, Clafer offers the best support.
Our analysis helps SPL users choose the tool that provides the best support according to the variability characteristics they need to model and the activities they need to carry out within an SPL. Moreover, tool developers can benefit from this analysis to focus on those activities that require better support.

6 SPL tools road map

This section answers our fourth research question:

RQ4: *Is it possible to carry out an SPL process, which includes the SPL activities and characteristics identified in the case studies analyzed, with the existing tool support?* **Rationale:** Even though the analyzed tools support some of the identified SPL activities and characteristics, not all tools provide support for the same activities and characteristics or in the same way, as demonstrated in Sect. 5. Moreover, the usage of a unique tool in isolation is not enough to support the complete process of an SPL that includes the four phases: DA, RA, DI, and PD, and thus, to support an SPL process completely, more than one tool needs to be employed depending on the specific requirements of the user. *We would like to know if a complete SPL process, including its four phases (DA, RA, DI, and PD), can be performed with the existing tools providing support for the different activities of those phases.*

To answer RQ4, based on the analysis in the previous section, we define some practical road map to completely carry out an SPL process with the existing tool support (Figure 5). The road map[15] defined in Figure 5 shows, for each phase and activity of the SPL, the recommended tool to be used. For example, the road map defined with **FeatureIDE** and **pure::variants** allow carrying out a complete SPL approach, covering all the activities of an SPL process and generating a final product. However, the limitations of these tools, as evidenced in Sect. 5, make them not suitable for complex domains, such as robotics or video systems, that demand advanced SPL characteristics such as clonable features, binding modes, multi-dimensional variability, managing large models, or dealing with NFPs.

To partly solve these issues, SPL users can combine some of the tools or integrate them. Following with our road map (Figure 5), the possible combinations are represented by the sequence flow that connects each activity of the SPL and are tagged with the tool that provides support for that activity. When we are only interested in analyzing the SPL variability, we can opt to use only **Glencoe** (for basic variability modeling) or **FAMA** (for extended variability modeling) that are the tools with the best support for automated reasoning. When we need to generate a specific configuration (or a partial one) based on the requirements of the application, **S.P.L.O.T.** and **Glencoe** offer an excellent online service for feature-based interactive configuration. When all configurations need to be generated at the RA phase, or we need to generate a subset of all the possible configurations, or we want to optimize configurations, our best option is **Clafer**. For implementing the reusable artifacts of the SPL from scratch, that is, following a proactive and/or reactive approach to develop an SPL [114], **FeatureIDE** and **pure::variants** are the recommendable choices because they allow using several languages and variability approaches (FOP, AOP, annotations). For an extractive approach where the user starts with a collection of existing products [114], **pure::variants** with its family model, which connects the existing artifacts with the FM, and **vEXgine**, which follows an orthogonal approach to define the variability [19], are good choices. For those domains (e.g., web engineering) in which applications require the combination of more than one different approach, users will need to implement specific composers to perform the combination work, like a new composer plugin for **FeatureIDE**. In this sense, **vEXgine** provides great flexibility because it is designed to be extensible by means of model transformations. Finally, to deal with variability models at the architectural level, **pure::variants** is the most mature tool, with the only drawback that the commercial version of the tool is required [86]. Also, **vEXgine** provides excellent support for resolving the variability of architectural models, but in this case the downside is that users need to

[15] We use the Business Process Model and Notation (BPMN) to represent the road map.

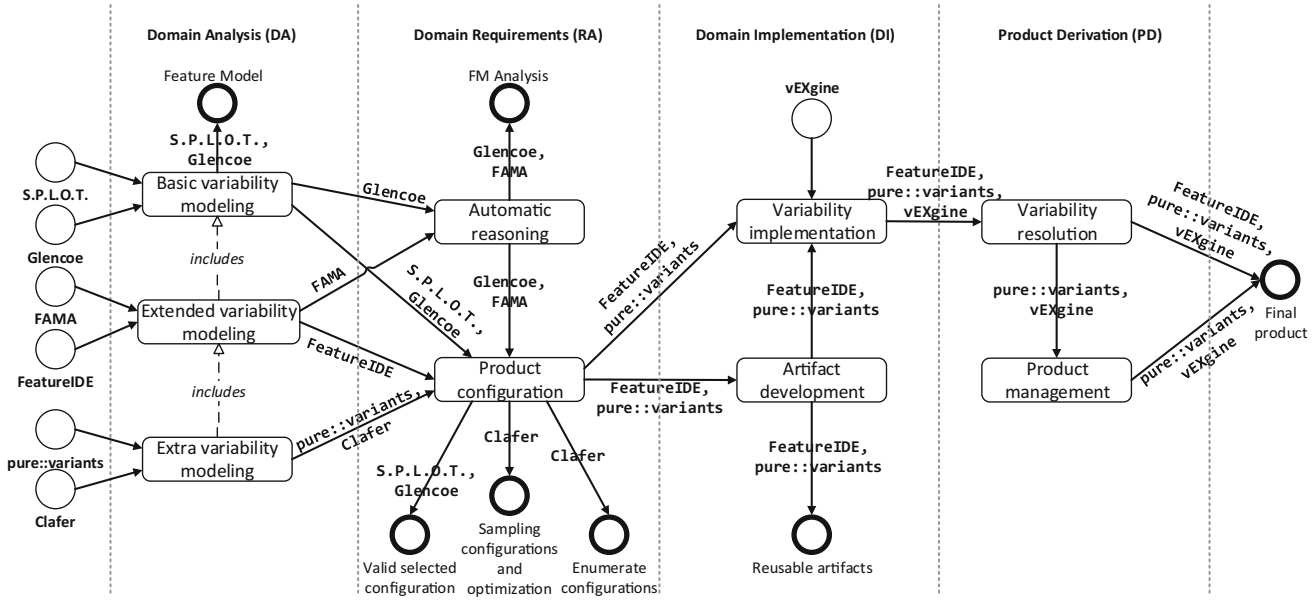

Fig. 5 Road map with the recommended tools for each phase and activity of an SPL

have some expertise in Model-Driven Engineering to define the appropriate model transformations.

We illustrate the usage of our road map with the following interoperability scenario. Let us suppose that a user Joseph needs to model the variability of an edge computing application [115], analyze its variability, and sample some valid configurations that optimize the system's performance to generate the final product. He decides to use the **S.P.L.O.T.** tool to specify the variability model using an online and easy-to-use web application. To automatically reason about the system's variability, the user exports the variability model into the **FAMA** tool, which provides good support for validity checking and finding inconsistencies. However, he realizes that none of these tools support all the variability characteristics required by the edge computing domain. As the other domains analyzed in Sect. 3, edge computing applications require the modeling of numerical features, clonable features, and complex cross-tree constraints involving some numerical values. To work with that "extra variability," the user exports its model to the **Clafer** tool, which allows modeling the numerical features and optimizing a configuration sample. Knowing the configurations that will be deployed, the SPL user needs to implement the variable artifacts and resolve the variability according to those configurations to generate the final product. To do that, he exports its model again to the **FeatureIDE** tool and implements the artifacts using the high diversity of variability implementation techniques offered by **FeatureIDE**. Finally, with the configurations previously identified, he generates the final products also using **FeatureIDE**. This chain of tools is possible because all these tools use interchangeable formats easy to import and export. **FeatureIDE**, **FAMA**, and **Clafer** support importing SXFM models from **S.P.L.O.T.** Similarly, **Glencoe** allows exporting the models to several formats (DIMACS, SPASS, v.control…) including the format used by **pure::variants**. To cover some of the possible connections in the road map (e.g., connect **S.P.L.O.T.** and **Clafer**), we have implemented the necessary scripts and algorithms, which are available online.[16]

[16] https://github.com/jmhorcas/SPLE-EmpiricalAnalysis.

Conclusions and lessons learned from RQ4: Existing tools support the complete process of SPL but with many limitations when dealing with complex variability requirements, demanding the usage of more than one tool. Concretely, for the DA phase, Clafer and pure::variants are the tools supporting more advanced mechanisms to model variability. However, for the RA phase, Glencoe and FAMA provide better support for automatic reasoning on those models, even though they do not support the advanced variability mechanisms completely. In the RA phase, Clafer can also be used for specific analysis operations such as the enumeration, sampling, and optimization of configurations despite its poor performance. Finally, FeatureIDE and pure::variants are the most appropriate tools for the DI and PD phases to support the implementation and resolution of the variability and the subsequent generation of the final product. Our road map will help SPL engineers to be aware of which tools can be used in isolation or in combination when a single tool does not support the complete SPL process.

7 Threats to validity

This section discusses the threats to validity of our study [116]:

Internal validity. An internal validity concern is the reliability of the experiments to check the *functionality fulfillment* of tools.

Functionality fulfillment. The functionality and characteristics analyzed vary among the tools. For example, clonable features are implemented differently in each tool. Literature reviews about tools usually study the support of functionalities as a primary goal. However, the goal of this paper is verifying how the tools satisfy the requirements in which we are interested to carry out a complex SPL process instead of reviewing all the available functionalities provided by the tools.

External validity. An external validity concerns the *generalization of the SPL and variability requirements* to others case studies and domains, beyond those discussed in Sect. 3.

Generalization of the requirements. We have especially looked for case studies that pose the most challenging requirements in the context of SPLs and variability modeling. We have analyzed a sample of 20 case studies in six different domains. We consider that this sample is representative enough, and indeed there are many more case studies and domains that share the same requirements, for example, some of the case studies in the ESPLA catalog [117]. We believe that our analysis of case studies is representative for the domains. We also conjecture that case studies in other related domains, especially the current trending topic domains, such as Internet of Things (IoT), Cyber-Physical Systems, Edge Computing, and web engineering, will share many characteristics and requirements with the studied systems.

Construct validity. Construct validity relates to the completeness of our study, as well as any potential bias.

Important tools missing in the state of the art. The search for the tools information was conducted in several SLRs, proceedings of the most relevant conferences in SPL (e.g., SPLC) and variability (e.g., VaMoS), as well as in web search engines, and it was gathered through a data extraction form. We believe that we do not have omitted any relevant tools. However, since new tools are constantly appearing and evolving, we encourage SPL researchers to fill the information about any missing or new SPL tools in our form so that we can include them and continuously extend our study.

Tools selection for analysis. The defined inclusion and exclusion criteria to select the tools for our analysis can exclude some relevant tools (e.g., Gears). Our criteria focus especially on the availability and usability of the tools that we consider the first obstacle for an advanced analysis of the tools. Therefore, we did not consider for our detailed analysis those tools that are not available to be directly downloaded, require to pay a license, or have inadequate documentation because those tools cannot be analyzed before acquiring them (case of industrial tools) or require continuously contacting the developers to solve issues or errors when using the tools (case of obsoleted and not available tools). A threat to validity is that, for those tools that were not able to be installed or had some errors or lack of documentation that prevented us from testing them, we decided not to contact the authors for help, since we consider that regular users often do not make so.

Biased judgment selection and analysis. As the researchers involved in this study are active in the SPL research area, a validity problem could be the author's bias in the selection process of studies and tools. Regarding the sampling of case studies, authors have been working for years to contribute to the improvement in convenience modeling languages due to the shortcomings they have in modeling certain characteristics. Part of our previous work Horcas et al. [24] and the limitations found were the starting point for the detailed study that has been conducted in this article. In fact, the specific case study presented in Horcas et al. [24], *WeaFQAs*, was not considered in the sample because it represents a cross-cutting domain (e.g., quality attributes), and no other case studies were found in that domain in the pool of 477 articles. Finally, only 2 of the 20 studies analyzed were published by the authors of this article. Regarding the tool selection, the authors of this article have produced several tools in SPL (e.g., vEXgine, HADAS, Hydra, AO-ADL). Only vEXgine passed our inclusion/exclusion criteria and was considered

to be further analyzed. In addition, the decision to include vEXgine over other similar tools is threefold: (1) actually, it is the only available tool to provide support for CVL models [87]; (2) it is one of the few tools that work at the architectural level; and (3) it is very flexible to be extended or integrated within any other tool or approach. Despite those benefits, vEXgine also presents some limitations as discussed in Sect. 5.

Conclusion validity. Conclusion validity relates to the *reliability and robustness of our results*.

Interpretation of the analysis results. A potential threat to conclusion validity is the interpretation of the results extracted from the analyzed tools. It was not always obvious to state from the empirical experiments if the tools satisfy the exposed requirements completely or partially. To ensure the validity of our results, apart from the empirical experiments, we analyzed multiple data sources (e.g., tool documentation, reference papers, technical reports…). Moreover, the experiments were carried out at least by two primary authors that acted as reviewers of the results reported by the others. Considering a larger number of evaluators might have contributed to a more extensive experimentation and a higher precision of the results. These external researchers would have helped to cross-check our results.

8 Related work

SPL phases and activities have been widely studied by researchers, but unfortunately, there are few empirical studies covering the use of those SPL activities in practice with the existing tool support.

8.1 SPL phases and activities

Multiple reviews and surveys have been published covering different aspects of SPL engineering, such as the level of alignment in the topics covered by academia and industry [20], the level of tool support [118], or the most researched topics in SPL [6,56]. These studies help to identify the phases and activities of SPL engineering that deserve more attention in the SPL community. For instance, the survey by Rabiser et al. [6] states that architecting (i.e., working at the architectural level) is the dominating SPL topic, covered by 38% of surveyed papers.

Other studies focus on specific phases of the SPL. For example, Schobbens et al. [34] survey the different languages and notations for variability modeling, while Berger et al. [20] address the use of variability modeling notations, the scalability of industrial models, and SPL tools in industry, and Benavides [10] focuses on the modeling and analysis of variability and, in particular, on the automated reasoning on feature models [47], imposing new challenges to the existing development and analysis activities, as well as on the tool support. The automatic configuration of products has been widely studied in multiple works, from works covering the optimization and trade-off of NFPs [105] or modeling performance of highly configurable systems [119], to surveys and systematic literature reviews focused on semi-automatic configuration of extended product lines [120], which include scalability and performance concerns [77]. Covering the domain implementation phase, Apel et al. [9] explain in detail well-known variability implementation techniques (e.g., components and services, preprocessors, design patterns, feature-oriented programming, aspect-oriented programming, virtual separation of concerns, etc.). They also list tools that provide support for those techniques as well as for other activities, such as mapping features or traceability, but without further details, in contrast to our deep analysis in Sect. 5.

8.2 SPL requirements

Regarding the requirements of SPLs, most of the research literature on SPL usually provides only small examples [9], and thus, tools are usually built to support specific case studies or toy applications. While no work studies the practical support of the existing tools for case studies [121], we have presented in Sect. 3 a sample of 20 case studies with complex requirements for SPL as motivation for the analysis of the tools.

8.3 SPL tools analysis

Few works study the tool support for the SPL phases and activities [8,21–23,118]. They are systematic literature reviews, mapping studies, or surveys that are normally done only from the perspective of the documentation found for each tool and the characteristics listed and discussed in that documentation. In addition, most of the details about the tools are covered in gray literature, thesis, and websites, that are not usually considered as primary studies in SLRs. For example, Bashroush et al. [8] study general characteristics of SPL tools, such as the technology used in its implementation, or the notation (graphical or textual) used for variability modeling. Other similar but older studies are presented by Pereira et al. [22] and by Lisboa et al. [23]. In particular, there are also some other works analyzing directly SPL tools by testing their usability and applicability [18,122], but these works consider only two tools to be analyzed. A more recent work [21] presents a systematic mapping study with more than a hundred of variability tools and up to 11 capabilities that are missed by the industry in those tools. In contrast to our study, where we provide a deep analysis of how each tool supports different variability characteristics, Allian et al. [21] analyze the tools by conducting a survey with practitioners

from the industry to analyze the missing capabilities. Moreover, some of the capabilities are common to any type of tools and not specific to SPL, such as collaborative support, scalability, or integration with testing tools.

However, these kinds of studies are not enough to select the most appropriate tool to provide support for an SPL process. This is because only information about the high-level phases covered by each tool is provided, omitting the details about the specific topics covered in each phase. Moreover, the information is extracted from the tool documentation or a reference paper, and thus, these studies become outdated very soon because, in most of the cases, they are not trying directly with the tools, downloading, installing, and executing the tools or even checking their online availability—i.e., many of the tools included in existing studies are not available at all. There are even tools referenced in these papers that have never been implemented [22]. Commercial tools like Gears [123] and **`pure::variants`** [86] present the additional problem of the intellectual property protection of their technical details [8].

9 Conclusions and future work

We have presented a state of the art of the tools for SPL, focusing on their availability and usability. Based on this study, we have later empirically analyzed the most usable tools to check out the existence of enough mature tool support to cover the current variability and SPL requirements of case studies in different domains. We have also defined a road map of the recommended tools to partially or completely support SPL activities, from the variability modeling until the product derivation phase.

The conclusion is that we need an integrated approach with appropriate tool support that covers all activities/phases that are normally performed in complex SPLs. The main characteristics that the tools should support are: (1) modeling the variability of complex features (e.g., clonable features, variable features, composite features), (2) flexibility in the analysis of large feature models considering sampling and optimization of configurations (e.g., analysis of NFPs), and (3) combination of multiple variability approaches (FOP, AOP, annotations) since only a variability approach (e.g., FOP) is not enough for some domains like cyber-physical systems or robotics that could greatly benefit from the use of SPLs. Therefore, with the existing tool support, it is possible to carry out a simple SPL process, but the tools present several limitations when dealing with complex SPLs.

As future work, we plan to continue our study to incorporate updated or new tools that could appear and that can be integrated into our road map.[17] In parallel, we plan to provide support for advanced variability modeling characteristics either by integrating them into existing tools or by developing new tools if needed. This will allow completing and improving the presented road map and interconnect the existing tools.

[17] https://github.com/jmhorcas/SPLE-EmpiricalAnalysis.

Acknowledgements This work is supported by the projects *MEDEA* RTI2018-099213-B-I00 (co-financed by FEDER funds), *LEIA* UMA18-FEDERIA-157, *Rhea* P18-FR-1081 (MCI/AEI/FEDER, UE), *TASOVA* MCIU-AEI TIN2017-90644-REDT, and European Union's H2020 research and innovation program under grant agreement *DAEMON* 101017109.

Funding Open Access funding provided thanks to the CRUE-CSIC agreement with Springer Nature. Funding for open access charge: Universidad de Málaga / CBUA

References

1. Berger, T., Collet, P.: Usage scenarios for a common feature modeling language. In: Proceedings of the 23rd International Systems and Software Product Line Conference, ACM, New York, NY, USA, SPLC'19, vol B, pp. 174–181 (2019). https://doi.org/10.1145/3307630.3342403
2. García, S., Strüber, D., Brugali, D., Di Fava, A., Schillinger, P., Pelliccione, P., Berger, T.: Variability modeling of service robots: experiences and challenges. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), ACM, New York, NY, USA, VAMOS'19, pp. 8:1–8:6 (2019). https://doi.org/10.1145/3302333.3302350
3. Nadi, S., Krüger, S.: Variability modeling of cryptographic components: Clafer experience report. In: Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems, ACM, New York, NY, USA, VaMoS'16, pp. 105–112 (2016). https://doi.org/10.1145/2866614.2866629
4. Berger, T., She, S., Lotufo, R., Wasowski, A., Czarnecki, K.: A study of variability models and languages in the systems software domain. IEEE Trans. Softw. Eng. **39**(12), 1611–1640 (2013). https://doi.org/10.1109/TSE.2013.34
5. Alférez, M., Acher, M., Galindo, J.A., Baudry, B., Benavides, D.: Modeling variability in the video domain: language and experience report. Softw. Qual. J. **27**(1), 307–347 (2019). https://doi.org/10.1007/s11219-017-9400-8
6. Rabiser, R., Schmid, K., Becker, M., Botterweck, G., Galster, M., Groher, I., Weyns, D.: A study and comparison of industrial vs. academic software product line research published at SPLC. In: Proceedings of the 22nd International Conference on Systems and

Software Product Line (SPLC), pp. 14–24 (2018). https://doi.org/10.1145/3233027.3233028
7. Chacón-Luna, A.E., Gutiérrez, A.M., Galindo, J.A., Benavides, D.: Empirical software product line engineering: a systematic literature review. Inf. Softw. Technol. **128**, 106389 (2020). https://doi.org/10.1016/j.infsof.2020.106389
8. Bashroush, R., Garba, M., Rabiser, R., Groher, I., Botterweck, G.: CASE tool support for variability management in software product lines. ACM Comput. Surv. **50**(1), 14:1-14:45 (2017). https://doi.org/10.1145/3034827
9. Apel, S., Batory, D.S., Kästner, C., Saake, G.: Feature-Oriented Software Product Lines - Concepts and Implementation. Springer, Berlin (2013). https://doi.org/10.1007/978-3-642-37521-7
10. Benavides, D.: Variability modelling and analysis during 30 years. In: From Software Engineering to Formal Methods and Tools, and Back - Essays Dedicated to Stefania Gnesi on the Occasion of Her 65th Birthday, pp. 365–373 (2019). https://doi.org/10.1007/978-3-030-30985-5_21
11. Laguna, M.A., Crespo, Y.: A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Sci. Comput. Program. **78**(8), 1010–1034 (2013). https://doi.org/10.1016/j.scico.2012.05.003.special section on software evolution, adaptability, and maintenance & Special section on the Brazilian Symposium on Programming Languages
12. Benavides, D., Trinidad, P., Cortés, A.R.: Automated reasoning on feature models. In: 17th International Conference on Advanced Information Systems Engineering CAiSE, pp. 491–503 (2005). https://doi.org/10.1007/11431855_34
13. Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J.: Non-Functional Requirements in Software Engineering. International Series in Software Engineering, vol 5. Springer (2000). https://doi.org/10.1007/978-1-4615-5269-7
14. Thüm, T., Kästner, C., Benduhn, F., Meinicke, J., Saake, G., Leich, T.: Featureide: an extensible framework for feature-oriented software development. Sci. Comput. Program. **79**, 70–85 (2014). https://doi.org/10.1016/j.scico.2012.06.002. Experimental Software and Toolkits (EST 4): A special issue of the Workshop on Academic Software Development Tools and Techniques (WASDeTT-3 2010)
15. Prehofer, C.: Feature-oriented programming: A fresh look at objects. In: Akşit, M., Matsuoka, S. (eds.) ECOOP'97—Object-Oriented Programming, pp. 419–443. Springer, Berlin (1997)
16. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In: Akşit, M., Matsuoka, S. (eds.) ECOOP'97—Object-Oriented Programming, pp. 220–242. Springer, Heidelberg (1997)
17. Hunsen, C., Zhang, B., Siegmund, J., Kästner, C., Leßenich, O., Becker, M., Apel, S.: Preprocessor-based variability in open-source and industrial software systems: an empirical study. Empir. Softw. Eng. **21**(2), 449–482 (2016). https://doi.org/10.1007/s10664-015-9360-1
18. Constantino, K., Pereira, J.A., Padilha, J., Vasconcelos, P., Figueiredo, E.: An empirical study of two software product line tools. In: Proceedings of the 11th International Conference on Evaluation of Novel Software Approaches to Software Engineering, SCITEPRESS - Science and Technology Publications, Lda, Portugal, ENASE 2016, pp. 164–171 (2016). https://doi.org/10.5220/0005829801640171
19. Horcas, J.M., Cortiñas, A., Fuentes, L., Luaces, M.R.: Combining multiple granularity variability in a software product line approach for web engineering. Inf. Softw. Technol. **148**, 106910 (2022)
20. Berger, T., Rublack, R., Nair, D., Atlee, J.M., Becker, M., Czarnecki, K., Wasowski, A.: A survey of variability modeling in industrial practice. In: Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems, ACM, New York, NY, USA, VaMoS'13, pp. 7:1–7:8 (2013a). https://doi.org/10.1145/2430502.2430513
21. Allian, A.P., OliveiraJr, E., Capilla, R., Nakagawa, E.Y.: Have variability tools fulfilled the needs of the software industry? J. Univ. Comput. Sci. **26**(10), 1282–1311 (2020)
22. Pereira, J.A., Constantino, K., Figueiredo, E.: A systematic literature review of software product line management tools. In: Schaefer, I., Stamelos, I. (eds.) Software Reuse for Dynamic Systems in the Cloud and Beyond, pp. 73–89. Springer, Cham (2014)
23. Lisboa, L.B., Garcia, V.C., Lucrédio, D., de Almeida, E.S., de Lemos Meira, S.R., de Mattos Fortes, R.P.: A systematic review of domain analysis tools. Inf. Softw. Technol. **52**(1), 1–13 (2010)
24. Horcas, J., Pinto, M., Fuentes, L.: Software product line engineering: a practical experience. In: Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC), ACM, vol A, pp. 25:1–25:13 (2019). https://doi.org/10.1145/3336294.3336304
25. Horcas, J.M.: WeaFQAs: a software product line approach for customizing and weaving efficient functional quality attributes. PhD thesis, Universidad de Málaga (2018). https://hdl.handle.net/10630/17231
26. Pohl, K., Böckle, G., Linden, FJvd: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin (2005)
27. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Reengineering legacy applications into software product lines: a systematic mapping. Empir. Softw. Eng. **22**(6), 2972–3016 (2017). https://doi.org/10.1007/s10664-017-9499-z
28. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-oriented domain analysis (FODA) feasibility study, Tech. rep.. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst (1990)
29. Czarnecki, K.,, She, S.,, Wasowski, A.: Sample spaces and feature models: There and back again. In: International Conference on Software Product Lines, SPLC, pp. 22–31 (2008). https://doi.org/10.1109/SPLC.2008.49
30. González-Baixauli, B., do Prado Leite, J.C.S., Mylopoulos, J.: Visual variability analysis for goal models. In: 12th IEEE International Conference on Requirements Engineering (RE), pp. 198–207 (2004). https://doi.org/10.1109/RE.2004.56
31. Schmid, K., John, I.: A customizable approach to full lifecycle variability management. Sci. Comput. Program. **53**(3), 259–284 (2004). https://doi.org/10.1016/j.scico.2003.04.002
32. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Olsen, G.K., Svendsen, A.: Adding standardized variability to domain specific languages. In: 12th International Software Product Line Conference (SPLC), pp. 139–148 (2008). https://doi.org/10.1109/SPLC.2008.25
33. Haugen, Ø., Øgård, O.: BVR - better variability results. In: International Conference on System Analysis and Modeling: Models and Reusability, SAM, pp. 1–15 (2014). https://doi.org/10.1007/978-3-319-11743-0_1
34. Schobbens, P., Heymans, P., Trigaux, J., Bontemps, Y.: Generic semantics of feature diagrams. Comput. Netw. **51**(2), 456–479 (2007). https://doi.org/10.1016/j.comnet.2006.08.008
35. Capilla, R., Dueñas, J.C.: Modelling variability with features in distributed architectures. In: Revised Papers from the 4th International Workshop on Software Product-Family Engineering, Springer-Verlag, Berlin, Heidelberg, PFE'01, pp. 319-329 (2001)
36. Munoz, D., Oh, J., Pinto, M., Fuentes, L., Batory, D.S.: Uniform random sampling product configurations of feature models that have numerical features. In: Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC), vol. A, pp. 39:1–39:13 (2019). https://doi.org/10.1145/3336294.3336297

37. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Formalizing cardinality-based feature models and their specialization. Softw. Process: Improv. Pract. **10**(1), 7–29 (2005). https://doi.org/10.1002/spip.213
38. Quinton, C., Romero, D., Duchien, L.: Cardinality-based feature models with constraints: a pragmatic approach. In: 17th International Software Product Line Conference (SPLC), pp. 162–166 (2013). https://doi.org/10.1145/2491627.2491638
39. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: FORM: a feature-oriented reuse method with domain-specific reference architectures. Ann. Softw. Eng. **5**, 143–168 (1998). https://doi.org/10.1023/A:1018980625587
40. Schroeter, J., Lochau, M., Winkelmann, T.: Multi-perspectives on feature models. In: 15th International Conference on Model Driven Engineering Languages and Systems (MODELS), pp. 252–268 (2012). https://doi.org/10.1007/978-3-642-33666-9_17
41. Rosenmüller, M., Siegmund, N., Thüm, T., Saake, G.: Multi-dimensional variability modeling. In: Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS), ACM, New York, NY, USA, VaMoS'11, pp. 11–20 (2011). https://doi.org/10.1145/1944892.1944894
42. Rosenmüller, M., Siegmund, N.: Automating the configuration of multi software product lines. In: Fourth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), pp. 123–130 (2010). http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf
43. Urli, S., Blay-Fornarino, M., Collet, P., Mosser, S.: Using composite feature models to support agile software product line evolution. In: Proceedings of the 6th International Workshop on Models and Evolution (ME@MoDELS), pp. 21–26 (2012). https://doi.org/10.1145/2523599.2523604
44. Gámez, N., Fuentes, L.: Software product line evolution with cardinality-based feature models. In: 12th International Conference on Software Reuse (ICSR) - Top Productivity through Software Reuse, pp. 102–118. (2011). https://doi.org/10.1007/978-3-642-21347-2_9
45. Thüm, T., Batory, D.S., Kästner, C.: Reasoning about edits to feature models. In: International Conference on Software Engineering, ICSE, pp. 254–264 (2009). https://doi.org/10.1109/ICSE.2009.5070526
46. Bosch, J., Florijn, G., Greefhorst, D., Kuusela, J., Obbink, J.H., Pohl, K.: Variability issues in software product lines. In: Software Product-Family Engineering, PFE, pp. 13–21 (2001). https://doi.org/10.1007/3-540-47833-7_3
47. Galindo, J.A., Benavides, D., Trinidad, P., Gutiérrez-Fernández, A.M., Ruiz-Cortés, A.: Automated analysis of feature models: Quo vadis? Computing **101**(5), 387–433 (2019). https://doi.org/10.1007/s00607-018-0646-1
48. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: a literature review. Inf. Syst. **35**(6), 615–636 (2010). https://doi.org/10.1016/j.is.2010.01.001
49. White, J., Dougherty, B., Schmidt, D.C., Benavides, D.: Automated reasoning for multi-step feature model configuration problems. In: International Software Product Line Conference (SPLC), pp. 11–20. Carnegie Mellon University, Pittsburgh (2009)
50. Krüger, J., Pinnecke, M., Kenner, A., Kruczek, C., Benduhn, F., Leich, T., Saake, G.: Composing annotations without regret? practical experiences using featureC. Softw. Pract. Exp. **48**(3), 402–427 (2018). https://doi.org/10.1002/spe.2525
51. Díaz, J., Pérez, J., Alarcón, P.P., Garbajosa, J.: Agile product line engineering—a systematic literature review. Softw. Pract. Exp. **41**(8), 921–941 (2011). https://doi.org/10.1002/spe.1087
52. Assunção, W.K.G., Lopez-Herrejon, R.E., Linsbauer, L., Vergilio, S.R., Egyed, A.: Multi-objective reverse engineering of variability-safe feature models based on code dependencies of system variants. Empir. Softw. Eng. **22**(4), 1763–1794 (2017). https://doi.org/10.1007/s10664-016-9462-4
53. Horcas, J.M., Pinto, M., Fuentes, L.: An automatic process for weaving functional quality attributes using a software product line approach. J. Syst. Softw. **112**, 78–95 (2016). https://doi.org/10.1016/j.jss.2015.11.005
54. Marques, M., Simmonds, J., Rossel, P.O., Bastarrica, M.C.: Software product line evolution: a systematic literature review. Inf. Softw. Technol. **105**, 190–208 (2019)
55. Horcas, J.M., Pinto, M., Fuentes, L.: Product line architecture for automatic evolution of multi-tenant applications. In: 20th IEEE International Enterprise Distributed Object Computing Conference (EDOC), pp. 1–10 (2016). https://doi.org/10.1109/EDOC.2016.7579384
56. Raatikainen, M., Tiihonen, J., Männistö, T.: Software product lines and variability modeling: a tertiary study. J. Syst. Softw. **149**, 485–510 (2019)
57. Ralph, P., Baltes, S., Bianculli, D., Dittrich, Y., Felderer, M., Feldt, R., Filieri, A., Furia, C.A., Graziotin, D., He, P., Hoda, R., Juristo, N., Kitchenham, B., Robbes, R., Mendez, D., Molleri, J., Spinellis, D., Staron, M., Stol, K., Tamburri, D., Torchiano, M., Treude, C., Turhan, B., Vegas, S.: Acm sigsoft empirical standards (2020). arXiv: 2010.03525
58. Dumitrescu, C., Mazo, R., Salinesi, C., Dauron, A.: Bridging the gap between product lines and systems engineering: an experience in variability management for automotive model based systems engineering. In: 17th International Software Product Line Conference (SPLC), pp. 254–263 (2013). https://doi.org/10.1145/2491627.2491655
59. Ali, S., Arcaini, P., Hasuo, I., Ishikawa, F., Lee, N.Z.: Towards a framework for the analysis of multi-product lines in the automotive domain. In: Proceedings of the 13th International Workshop on Variability Modelling of Software-Intensive Systems, ACM, New York, NY, USA, VAMOS'19, pp. 12:1–12:6 (2019). https://doi.org/10.1145/3302333.3302345
60. Horcas, J.M., Monteil, J., Bouroche, M., Pinto, M., Fuentes, L., Clarke, S.: Context-dependent reconfiguration of autonomous vehicles in mixed traffic. J. Softw. Evol. Process **30**(4), 1926 (2018). https://doi.org/10.1002/smr.1926
61. Temple, P., Acher, M., Perrouin, G., Biggio, B., Jezequel, J.M., Roli, F.: Towards quality assurance of software product lines with adversarial configurations. In: Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'19, vol A, pp. 277–288 (2019). https://doi.org/10.1145/3336294.3336309
62. Acher, M., Collet, P., Lahire, P., France, R.B.: FAMILIAR: a domain-specific language for large scale management of feature models. Sci. Comput. Program. **78**(6), 657–681 (2013)
63. Acher, M., Collet, P., Lahire, P., Moisan, S., Rigault, J.: Modeling variability from requirements to runtime. In: 16th IEEE International Conference on Engineering of Complex Computer Systems, pp. 77–86 (2011). https://doi.org/10.1109/ICECCS.2011.15
64. Arzt, S., Nadi, S., Ali, K., Bodden, E., Erdweg, S., Mezini, M.: Towards secure integration of cryptographic software. In: ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!), ACM, New York, NY, USA, Onward! 2015, pp. 1–13 (2015). https://doi.org/10.1145/2814228.2814229
65. Horcas, J., Pinto, M., Fuentes, L.: Automatic enforcement of security properties. In: 13th International Conference on Trust, Privacy and Security in Digital Business (TrustBus), pp. 19–31 (2016a). https://doi.org/10.1007/978-3-319-44341-6_2
66. Berger, T., Pfeiffer, R., Tartler, R., Dienst, S., Czarnecki, K., Wasowski, A., She, S.: Variability mechanisms in software

ecosystems. Inf. Softw. Technol. **56**(11), 1520–1535 (2014). https://doi.org/10.1016/j.infsof.2014.05.005
67. Krüger, J., Nielebock, S., Krieter, S., Diedrich, C., Leich, T., Saake, G., Zug, S., Ortmeier, F.: Beyond software product lines: Variability modeling in cyber-physical systems. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'17, vol A, pp. 237–241 (2017). https://doi.org/10.1145/3106195.3106217
68. Arrieta, A., Sagardui, G., Etxeberria, L.: Cyber-physical systems product lines: Variability analysis and challenges. In: VI Jornadas de Computación Empotrada (2015)
69. Beek, M.H., Fantechi, A., Gnesi, S.: Product line models of large cyber-physical systems: The case of ERTMS/ETCS. In: Proceedings of the 22nd International Systems and Software Product Line Conference, ACM, New York, NY, USA, SPLC'18, vol 1, pp. 208–214 (2018). https://doi.org/10.1145/3233027.3233046
70. Abd Halim, S., Jawawi, D., Ibrahim, N., deris, S.: An approach for representing domain requirements and domain architecture in software product line. (2012). https://doi.org/10.5772/37704
71. Gherardi, L., Hunziker, D., Mohanarajah, G.: A software product line approach for configuring cloud robotics applications. In: Proceedings of the IEEE International Conference on Cloud Computing, IEEE Computer Society, Washington, DC, USA, CLOUD'14, pp. 745–752 (2014). https://doi.org/10.1109/CLOUD.2014.104
72. Ziadi, T., Farges, J.L., Stinckwich, S., Ziane, M., Dhouib, S., Marmoiton, F., Morette, N., Novales, C., Kchir, S., Patin, B.: A toolset to address variability in mobile robotics. J. Softw. Eng. Robot. **7**(1), 20–35 (2016). https://hal.sorbonne-universite.fr/hal-01353929
73. Kang, K.C., Kim, M., Lee, J., Kim, B.: Feature-oriented re-engineering of legacy systems into product line assets—a case study. In: Obbink, H., Pohl, K. (eds.) Software Product Lines, pp. 45–56. Springer, Heidelberg (2005)
74. Horcas, J.M., Pinto, M., Fuentes, L.: An aspect-oriented model transformation to weave security using CVL. In: Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD), pp. 138–147 (2014). https://doi.org/10.5220/0004890601380147
75. She, S., Lotufo, R., Berger, T., Wasowski, A., Czarnecki, K.: The variability model of the linux kernel. In: Benavides, D., Batory, D.S., Grünbacher, P. (eds.) Fourth International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS), Universität Duisburg-Essen, ICB-Research Report, vol. 37, pp. 45–51 (2010). http://www.vamos-workshop.net/proceedings/VaMoS_2010_Proceedings.pdf
76. Pett, T., Thüm, T., Runge, T., Krieter, S., Lochau, M., Schaefer, I.: Product sampling for product lines: the scalability challenge. In: Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC), ACM, vol A, pp. 14:1–14:6 (2019). https://doi.org/10.1145/3336294.3336322,
77. Ochoa, L., Pereira, J.A., González-Rojas, O., Castro, H., Saake, G.: A survey on scalability and performance concerns in extended product lines configuration. In: Proceedings of the Eleventh International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), ACM, New York, NY, USA, VAMOS'17, pp. 5–12 (2017). https://doi.org/10.1145/3023956.3023959
78. Mendonca, M., Branco, M., Cowan, D.: S.P.L.O.T: software product lines online tools. In: Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems Languages and Applications, ACM, New York, NY, USA, OOPSLA'09, pp. 761–762 (2009). https://doi.org/10.1145/1639950.1640002
79. Schmitt, A., Bettinger, C., Rock, G.: Glencoe—a tool for specification, visualization and formal analysis of product lines. In: Transdisciplinary Engineering Methods for Social Innovation of Industry 4.0, Advances in Transdisciplinary Engineering, vol. 7, pp. 665–673 (2018). https://doi.org/10.3233/978-1-61499-898-3-665
80. Spinczyk, O., Beuche, D.: Modeling and building software product lines with eclipse. In: Companion to the 19th Annual ACM SIGPLAN Conference on Object-oriented Programming Systems, Languages, and Applications, ACM, New York, NY, USA, OOPSLA'04, pp. 18–19 (2004). https://doi.org/10.1145/1028664.1028675
81. Antkiewicz, M., Bak, K., Murashkin, A., Olaechea, R., Liang, J.H.J., Czarnecki, K.: Clafer tools for product line engineering. In: Proceedings of the 17th International Software Product Line Conference Co-located Workshops, ACM, New York, NY, USA, SPLC'13 Workshops, pp. 130–135 (2013). https://doi.org/10.1145/2499777.2499779
82. Prud'homme, C., Fages, J.G., Lorca, X.: Choco Documentation. TASC - LS2N CNRS UMR 6241, COSLING S.A.S., (2017). http://www.choco-solver.org
83. Benavides, D., Segura, S., Trinidad, P., Cortés, A.R.: FAMA: tooling a framework for the automated analysis of feature models. In: International Workshop on Variability Modelling of Software-Intensive Systems, VaMoS, pp. 129–134 (2007)
84. Leich, T., Apel, S., Marnitz, L., Saake, G.: Tool support for feature-oriented software development: FeatureIDE: an eclipse-based approach. In: Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange, ACM, New York, NY, USA, eclipse'05, pp. 55–59 (2005). https://doi.org/10.1145/1117696.1117708
85. Krieter, S., Pinnecke, M., Krüger, J., Sprey, J., Sontag, C., Thüm, T., Leich, T., Saake, G.: FeatureIDE: Empowering third-party developers. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'17, vol. B, pp. 42–45 (2017). https://doi.org/10.1145/3109729.3109751
86. Beuche, D.: Using pure: variants across the product line lifecycle. In: Proceedings of the 20th International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'16, pp. 333–336 (2016). https://doi.org/10.1145/2934466.2962729
87. Horcas, J.M., Pinto, M., Fuentes, L.: Extending the common variability language (cvl) engine: a practical tool. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'17, vol. B, pp. 32–37 (2017). https://doi.org/10.1145/3109729.3109749
88. Yli-Huumo, J., Maglyas, A., Smolander, K.: How do software development teams manage technical debt? An empirical study. J. Syst. Softw. **120**, 195–218 (2016). https://doi.org/10.1016/j.jss.2016.05.018
89. Apel, S., Kästner, C., Lengauer, C.: Language-independent and automated software composition: the featurehouse experience. IEEE Trans. Softw. Eng. **39**(1), 63–79 (2013). https://doi.org/10.1109/TSE.2011.120
90. Batory, D., Sarvela, J.N., Rauschmayer, A.: Scaling step-wise refinement. IEEE Trans. Softw. Eng. **30**(6), 355–371 (2004). https://doi.org/10.1109/TSE.2004.23
91. Horcas, J.M., Pinto, M., Fuentes, L.: Variability and dependency modeling of quality attributes. In: 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp 185–188 (2013). https://doi.org/10.1109/SEAA.2013.20
92. Ghofrani, J., Fehlhaber, A.L.: Productlinre: Online management tool for requirements engineering of software product lines. In: Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'18, vol. 2, pp 17–22 (2018). https://doi.org/10.1145/3236405.3236407

93. Basile, D., Di Giandomenico, F., Gnesi, S.: FMCAT: Supporting dynamic service-based product lines. In: Proceedings of the 21st International Systems and Software Product Line Conference - Volume B, ACM, New York, NY, USA, SPLC'17, pp. 3–8 (2017). https://doi.org/10.1145/3109729.3109760
94. Munoz, D., Pinto, M., Fuentes, L.: Finding correlations of features affecting energy consumption and performance of web servers using the HADAS eco-assistant. Computing **100**(11), 1155–1173 (2018). https://doi.org/10.1007/s00607-018-0632-7
95. Czarnecki, K.: Generative programming: Methods, techniques, and applications tutorial abstract. In: Gacek, C. (ed.) Software Reuse: Methods, Techniques, and Tools, pp. 351–352. Springer, Heidelberg (2002)
96. Svendsen, A., Zhang, X., Fleurey, F., Haugen, Ø., Olsen, G.K., Møller-Pedersen, B.: CVL tool - modeling variability in spls. In: 14th International Conference Software Product Lines (SPLC), vol 2, p. 299 (2010)
97. Vasilevskiy, A., Haugen Chauvel, F., Johansen, M.F., Shimbara, D.: The bvr tool bundle to support product line engineering. In: Proceedings of the 19th International Conference on Software Product Line (SPLC), ACM, New York, NY, USA, SPLC'15, pp. 380–384 (2015). https://doi.org/10.1145/2791060.2791094
98. Cordy, M., Schobbens, P.Y., Heymans, P., Legay, A.: Beyond boolean product-line model checking: Dealing with feature attributes and multi-features. In: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, Piscataway, NJ, USA, ICSE'13, pp. 472–481 (2013). http://dl.acm.org/citation.cfm?id=2486788.2486851
99. Czarnecki, K., Wasowski, A.: Feature diagrams and logics: There and back again. In: 11th International Software Product Line Conference (SPLC), pp. 23–34 (2007). https://doi.org/10.1109/SPLINE.2007.24
100. Heradio, R., Fernández-Amorós, D., Galindo, J.A., Benavides, D., Batory, D.S.: Uniform and scalable sampling of highly configurable systems. Empir. Softw. Eng. **27**(2), 44 (2022). https://doi.org/10.1007/s10664-021-10102-5
101. Oztok, U., Darwiche, A.: A top-down compiler for sentential decision diagrams. In: Proceedings of the 24th International Conference on Artificial Intelligence. AAAI Press, IJCAI'15, pp. 3141–3148 (2015). http://dl.acm.org/citation.cfm?id=2832581.2832687
102. Pereira, J.A., Matuszyk, P., Krieter, S., Spiliopoulou, M., Saake, G.: Personalized recommender systems for product-line configuration processes. Comput. Lang. Syst. Struct. **54**, 451–471 (2018). https://doi.org/10.1016/j.cl.2018.01.003
103. Pereira, J.A., Martinez, J., Gurudu, H.K., Krieter, S., Saake, G.: Visual guidance for product line configuration using recommendations and non-functional properties. In: Haddad, H.M., Wainwright, R.L., Chbeir, R. (eds.) Proceedings of the 33rd Annual ACM Symposium on Applied Computing (SAC), ACM, pp. 2058–2065 (2018). https://doi.org/10.1145/3167132.3167353
104. Horcas, J.M., Pinto, M., Fuentes, L.: Variability models for generating efficient configurations of functional quality attributes. Inf. Softw. Technol. **95**, 147–164 (2018). https://doi.org/10.1016/j.infsof.2017.10.018
105. Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., Saake, G.: SPL conqueror: toward optimization of non-functional properties in software product lines. Softw. Qual. J. **20**(3–4), 487–517 (2012). https://doi.org/10.1007/s11219-011-9152-9
106. Oh, J., Batory, D., Myers, M., Siegmund, N.: Finding near-optimal configurations in product lines by random sampling. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ACM, New York, NY, USA, ESEC/FSE 2017, pp. 61–71 (2017). https://doi.org/10.1145/3106237.3106273
107. Xiang, Y., Zhou, Y., Zheng, Z., Li, M.: Configuring software product lines by combining many-objective optimization and sat solvers. ACM Trans. Softw. Eng. Methodol. **26**(4), 14:1-14:46 (2018). https://doi.org/10.1145/3176644
108. Buccella, A., Pol'la, M., de Galarreta, E.R., Cechich, A.: Combining automatic variability analysis tools: an SPL approach for building a framework for composition. In: Gervasi, O., Murgante, B., Misra, S., Stankova, E., Torre, C.M., Rocha, A.M.A., Taniar, D., Apduhan, B.O., Tarantino, E., Ryu, Y. (eds.) Computational Science and Its Applications - ICCSA 2018, pp. 435–451. Springer, Cham (2018)
109. Meinicke, J., Thüm, T., Schröter, R., Benduhn, F., Leich, T., Saake, G.: Mastering Software Variability with FeatureIDE. Springer, Berlin (2017). https://doi.org/10.1007/978-3-319-61443-4
110. Schulze, S., Schulze, M., Ryssel, U., Seidl, C.: Aligning coevolving artifacts between software product lines and products. In: Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS), ACM, New York, NY, USA, VaMoS'16, pp. 9–16 (2016). https://doi.org/10.1145/2866614.2866616
111. Narwane, G.K., Galindo, J.A., Krishna, S.N., Benavides, D., Millo, J.V., Ramesh, S.: Traceability analyses between features and assets in software product lines. Entropy **18**, 269 (2016)
112. Hellebrand, R., Schulze, M., Ryssel, U.: Reverse engineering challenges of the feedback scenario in co-evolving product lines. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC '17, vol B, pp. 53–56 (2017). https://doi.org/10.1145/3109729.3109735
113. Ayala, I., Amor, M., Horcas, J.M., Fuentes, L.: A goal-driven software product line approach for evolving multi-agent systems in the internet of things. Knowl. Based Syst. **184**, 104883 (2019). https://doi.org/10.1016/j.knosys.2019.104883
114. Clements, P.C., Krueger, C.W.: Point—counterpoint: being proactive pays off—eliminating the adoption. IEEE Softw. **19**(4), 28–31 (2002). https://doi.org/10.1109/MS.2002.1020283
115. Cañete, A., Amor, M., Fuentes, L.: Supporting IoT applications deployment on edge-based infrastructures using multi-layer feature models. J. Syst. Softw. **183**, 111086 (2022). https://doi.org/10.1016/j.jss.2021.111086
116. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B.: Experimentation in Software Engineering. Springer, Berlin (2012). https://doi.org/10.1007/978-3-642-29044-2
117. Martinez, J., Assunção, W.K.G., Ziadi, T.: ESPLA: a catalog of extractive SPL adoption case studies. In: Proceedings of the 21st International Systems and Software Product Line Conference (SPLC), ACM, vol. B, pp. 38–41 (2017). https://doi.org/10.1145/3109729.3109748
118. Capilla, R., Sánchez, A., Dueñas, J.C.: An analysis of variability modeling and management tools for product line development. In: Software and Service Variability Management Workshop-Concepts, Models, and Tools, pp. 32–47 (2007)
119. Kolesnikov, S.S., Siegmund, N., Kästner, C., Grebhahn, A., Apel, S.: Tradeoffs in modeling performance of highly configurable software systems. Softw. Syst. Model. **18**(3), 2265–2283 (2019). https://doi.org/10.1007/s10270-018-0662-9
120. Ochoa, L., Rojas, O.G., Pereira, J.A., Castro, H., Saake, G.: A systematic literature review on the semi-automatic configuration of extended product lines. J. Syst. Softw. **144**, 511–532 (2018). https://doi.org/10.1016/j.jss.2018.07.054
121. Tolvanen, J.P., Kelly, S.: How domain-specific modeling languages address variability in product line development: investigation of 23 cases. In: Proceedings of the 23rd International Systems and Software Product Line Conference (SPLC), ACM, New York,

NY, USA, SPLC'19, vol. A, pp. 155–163 (2019). https://doi.org/10.1145/3336294.3336316
122. Pereira, J.A., Souza, C., Figueiredo, E., Abílio, R., Vale, G., Costa, H.A.X.: Software variability management: an exploratory study with two feature modeling tools. In: VII Brazilian Symposium on Software Components, Architectures and Reuse, IEEE Computer Society, pp. 20–29 (2013). https://doi.org/10.1109/SBCARS.2013.13
123. Krueger, C., Clements, P.: Feature-based systems and software product line engineering with gears from biglever. In: Proceedings of the 22nd International Systems and Software Product Line Conference (SPLC), ACM, New York, NY, USA, SPLC'18, vol. 2, pp. 1–4 (2018). https://doi.org/10.1145/3236405.3236409

José Miguel Horcas is a postdoc researcher at the University of Málaga, Spain, where he received his M.Sc. degree in computer science in 2012 and the Ph.D. degree in 2018. He is a member of the CAOSD research group and has carried out two postdoct stays at King's College London in 2019 for three months and at the University of Seville for 18 months. His main research areas are related to software product lines, including variability and configurability, and quality attributes. More information available at https://sites.google.com/view/josemiguelhorcas

.

Mónica Pinto received the M.Sc. degree in computer science and the Ph.D. degree from the Universidad de Málaga, Spain, in 1998 and 2004, respectively. She is an Associate Professor since 2009 with the Department of Lenguajes y Ciencias de la Computación, Universidad de Málaga. She is a research member of the CAOSD research group, one of the constituent group of the Instituto de Tecnología e Ingeniería del Software "José María Troya Linero of the Universidad de Málaga. She is currently part of the institute management team. She actively participates in Spanish and European research projects. Her main research areas are energy-aware software development, quality-driven variability modeling and analysis, model-driven software engineering, and Internet Of Things and Edge computing systems development.

Lidia Fuentes received the M.Sc. and Ph.D. degrees in computer science from the Universidad de Málaga, Spain, in 1998. She has done all her teaching work with the Department Lenguajes y Ciencias de la Computación, Universidad de Málaga since 1993, being the first female Full Professor, where she is the Head of the CAOSD Research Group. She has coauthored more than 200 publications in software engineering techniques applied to IoT and cyber-physical systems. Dr. Fuentes has an important international profile leading European projects and as a member of program committees of prestigious international conferences. Examples are ECOOP, SPLC, Modularity/AOSD, and OOPSLA.