Software Product Line Engineering: A Practical Experience

Jose-Miguel Horcas
horcas@lcc.uma.es
Univ. de Málaga, CAOSD group, Spain

Mónica Pinto
pinto@lcc.uma.es
Univ. de Málaga, CAOSD group, Spain

Lidia Fuentes
lff@lcc.uma.es
Univ. de Málaga, CAOSD group, Spain

ABSTRACT

The lack of mature tool support is one of the main reasons that make the industry to be reluctant to adopt Software Product Line (SPL) approaches. A number of systematic literature reviews exist that identify the main characteristics offered by existing tools and the SPL phases in which they can be applied. However, these reviews do not really help to understand if those tools are offering what is really needed to apply SPLs to complex projects. These studies are mainly based on information extracted from the tool documentation or published papers. In this paper, we follow a different approach, in which we firstly identify those characteristics that are currently essential for the development of an SPL, and secondly analyze whether the tools provide or not support for those characteristics. We focus on those tools that satisfy certain selection criteria (e.g., they can be downloaded and are ready to be used). The paper presents a state of practice with the availability and usability of the existing tools for SPL, and defines different roadmaps that allow carrying out a complete SPL process with the existing tool support.

KEYWORDS

SPL in practice, state of practice, tool support, tooling roadmap

1 INTRODUCTION

An increasing number of software development companies are adopting *Software Product Line* (SPL) approaches [8, 47]. However, the field of SPL is quite broad and continuously changing [57]. Technical issues such as the development of the SPL infrastructure, including new practices, processes, and tool support require a great investment [8]. Thus, despite the amount of successful stories about the use of SPL engineering[1], industry has not yet solved the variability and reuse management problem and continues to experiment with their own solutions and approaches [13]. This is partly due to the fact that companies are reluctant to adopt technically sound academic approaches due to the lack of mature tool support. Hence, the success of SPL depends on good *tool support* as much as on complete *SPL engineering processes* [6].

Regarding the processes, SPL approaches typically cover the *domain* and *application engineering* processes that include activities such as variability modeling and artifact implementation (domain engineering) and requirements analysis and product derivation (application engineering) [4, 11], but set aside other activities that are important for companies, such as the analysis of *non-functional properties* (NFPs) or *quality attributes* and the *evolution* of SPL's artifacts [41]. When considered, these activities are usually integrated into the traditional SPL process by reusing existing mechanisms, which were not specifically designed for that purpose (e.g., using attributes of extended variability models to model NFPs [10]). The most common is to find companies that only adopt a minor part of an SPL methodology (e.g., implementing variability with annotations) [30], which is sometime abandoned after a short period of time because of the lack of integration among the SPL activities. Only a few companies with enough financial resources have succeeded on using SPLs, due in most cases to the development of their own tools or the use of commercial tools (e.g., pure::variants [12], Gears [40]) that are not so accessible for smaller companies.

Besides, although tool support is of paramount importance for the SPL management process [6], most existing tools only cover specific phases of the SPL approach (e.g., variability modeling or artifacts implementation). Those few tools that support several phases (e.g., FeatureIDE [64]) demand to fit an implementation technique such as *Feature-Oriented Programming* (FOP) [53], *Aspect-Oriented Programming* (AOP) [38] or *annotations* [37], depend on the development IDE (e.g., Eclipse) or present some important limitations [20]. For instance, applying classical SPL approaches (e.g., FOP or AOP) to web engineering is challenging because of the nature of web applications that require the simultaneous use of several languages (JavaScript, Python, Groovy,...) in the same application [30]. In addition, there are few works specifically focused on studying the SPL tool support [6, 43, 51] and, they usually report information that is extracted from the tool documentation or reference papers without really testing the tool *availability* and *usability*.

This paper explores the existing tool support for SPL from a practical point of view, although it does not pretend to be a systematic review. The objective is to check out the existence of enough mature tool support for carrying out a complete SPL process. For each activity in the domain and application engineering, we identify the desired requirements that tools should provide to deal with complex SPL processes and application domains, and analyze the

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

possibilities and limitations of each tool. The paper answers the following Research Questions:

RQ1: *What are the tools that provide some kind of support for SPLs? Are they available, usable, and keep up to date?* To answer to this question, this paper presents a *state of practice* of the SPL tools, focusing on their availability and usability and selecting those that a company could use in its development process to successfully adopt an SPL process (Section 4).

RQ2: *How the tools behave when they are used to develop complex SPLs? (Either because a specific SPL development approach is required or because SPLs for complex application domains need to be developed.)* We answer to this question by *empirically analyzing* the most usable tools. We use a running case study based on an SPL process with demanding characteristics such as clonable features [24, 60], variable features [21, 22], attributes [10], huge feature models, and so on (Section 5).

RQ3: *Is it possible to carry out a complete SPL process with the existing tool support? That is, is it possible to cover all activities of complex approaches including dealing with NFPs, and managing the evolution of SPLs?* We answer to this question by defining up to 12 different *roadmaps* of tools that partially or completely support all phases of an SPL process (Section 6).

The paper is structured as follows. Section 2 discusses related work. Section 3 motivates our study showing the requirements of complex SPL approaches. Section 4 presents a state of practice of the existing tools. Section 5 analyzes the most usable tools. Section 6 defines the tool roadmaps to carry out a complete SPL. Section 7 discusses threats to validity and Section 8 concludes the paper.

2 RELATED WORK

Existing works have investigated the SPL processes in great detail [17, 56, 57], but has only surfaced its tool support [6, 43, 51].

2.1 Software Product Line processes

Multiple systematic literature reviews (SLRs) and surveys have been published covering different aspects of SPL engineering [11, 16, 17, 48, 56, 57], such as the level of alignment in the topics covered by academia and industry [11], the level of tool support [16] or the most researched topics in SPL [56, 57]. From these studies the phases and topics of SPL engineering in which academia and industry are more interested or that deserve more attention can be identified. These SLRs highlight some interesting conclusions. For instance, architecting [15] is the dominating SPL activity, covered by 38% of surveyed papers (56% of them are industry papers) [57].

However, there are other activities that are becoming important in the context of SPL with the emerging of new application domains, such as the Internet of Things (IoT) [55] or Cyber-Physical Systems [35], and that are not receiving the required attention. Examples of these activities are the optimization of large-scale variability models [48, 49], the variability modeling of quality attributes [31, 68], the management of NFPs [34, 48], and the evolution of the SPL models [32, 41]. This imposes new challenges to the existing development and analyses processes, as well as to the tool support. Even so, as exposed by those SLRs [11, 57], the variability in quality attributes is a concern only in 6% of the papers, and NFPs are discussed only in 5% of all the papers [57].

2.2 Software Product Line tools

Few works study the tool support for the SPL processes specifically [6, 16, 43, 51]. They are SLRs 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. Moreover, most of the details about tools are covered in gray literature, thesis, and websites, that are not usually considered as primary studies in SLRs. Commercial tools (e.g., Gears [40] and pure::variants [12]) present the additional problem of the intellectual property protection of their technical details [6]. Concretely, in [12], a demonstration of pure::variants across the product line lifecycle is described, but it only surfaces the tool without providing further technical details. The same occurs with Gears in [40].

One of the most recent systematic studies [6] covers tools documented in research papers from 1997 until 2015 (although the study was published in 2017). This is an interesting study to know the general characteristics of these tools (e.g., technology used in its implementation, if it has a graphical or textual notation, etc.). Others similar, but older studies are [51] (published in 2014) and [43] (published in 2010). However, the most recent tools reported by these studies date from 2013 and 2005, respectively.

The conclusion is that these kinds of studies are not enough to select the most appropriate tool to provide support for an SPL process. This is basically because only information about the high-level phases covered by each tool is provided, omitting the details about the specific topics covered for each phase. In addition, the information is extracted from the tool documentation or a reference paper, and thus, these studies stay outdated very soon because, in most of the cases, they are not trying and striving 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 [51].

3 MOTIVATION AND CASE STUDY

An SPL approach should cover at least the domain engineering and the application engineering processes with their typical phases and activities (Figure 1): (1) variability and dependency modeling in the Domain Analysis (DA) phase; (2) feature selection and product configuration in the Requirements Analysis (RA) phase; (3) the development of reusable software artifacts in the Domain Implementation (DI) phase; and (4) variability resolution and product generation in the Product Derivation (PD) phase [4]. However, demanding requirements of real SPL projects expose the needs of additional activities that are present in any software engineering process and that are essentials for the successful adoption of SPLs in industry. Some examples are the analysis of NFPs or quality attributes, or the evolution of the SPL's artifacts, among others.

To illustrate those points, we present *WeaFQAs* [29, 31, 34], an example of an SPL process that extends the engineering framework for SPL [52] and demands specific needs in each of the SPL phases according to the current applications and domains where WeaFQAs may be used. In this section, we describe the particular needs that WeaFQAs imposes in each phase of the classical SPL framework and enumerate the requirements that tools should satisfy.

3.1 Case Study: the WeaFQAs SPL process

WeaFQAs [29, 31, 34] is an SPL process to manage the *operationalizations* of quality attributes. The operationalization of a quality attribute (e.g., security) is the association of a function (e.g., the encryption of a message) to a goal (e.g., providing security). WeaFQAs introduces the concept of *Functional Quality Attribute* (FQA) as the specific functionality that is incorporated into the applications to fulfill the desired quality attribute. WeaFQAs promotes the variability modeling and customization of FQAs separately from the applications, and their later incorporation into them, exploiting the benefits of SPLs (e.g., reusability, adaptability,. . .). WeaFQAs follows the classic framework for SPL engineering [52] (Figure 1) — i.e., DA, RA, DI, and PD., but needs to extend it to take into account the specific problematic of FQAs, which requires additional activities in each of those phases (see activities in bold in Figure 1).

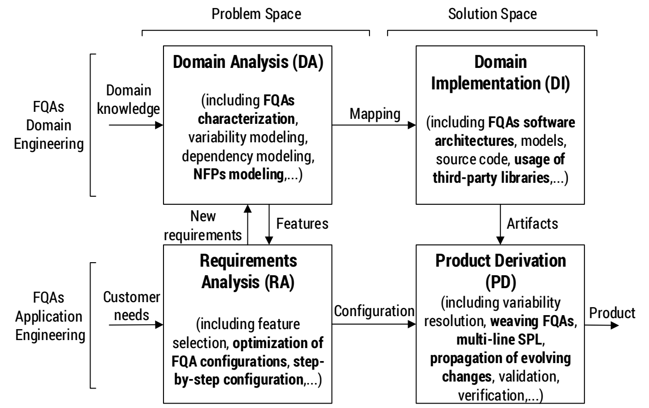

Figure 1: The WeaFQAs SPL process.

3.2 Requirements of complex SPLs

Domain Analysis (DA). Modeling a family of FQAs starts with the characterization of the FQAs and their variability modeling (Figure 2). For each quality attribute (e.g., security) the FQAs that are required to satisfy it are modeled (e.g, encryption, authentication). For instance, characterizing the encryption FQA includes the identification of the different encryption algorithms, the available security frameworks that implement it, along with their variables and parameters such as the key length, block cipher mode of operation, and padding; the dependency relationships between encryption and others FQAs (e.g., hashing); as well as the usage context [34].

Nevertheless, modeling the FQAs variability is more complex that only considering their high degree of variability. Since each FQA (e.g., encryption) can be applied in several points of the application with different configurations (e.g., RSA or AES algorithms), each FQA needs to be modeled as a *clonable feature* [24, 60] (see `Encryption[1..*]` in Figure 2). Moreover, some FQAs include variables that require to provide specific values (e.g., 512 kB as the average size of the messages to be encrypted). This requires to model them as *variable features* [21, 22] (see `MessageSize` in Figure 2). Here, it is worthy to differentiate between variable features [21, 22] that are those that require to provide a value (e.g., integer, string, float) during configuration; and *features with attributes* [10], which can be used to model NFPs such as the performance or price of features (see `Execution time` and `Energy consumption` attributes of the `RSA` feature in Figure 2). The incorporation of clonable and variable features brings into play the definition of complex cross-tree constraints. Examples are modeling the dependency between features that are children of different clonable features, or a dependency involving a numerical value (see cross-tree constraints in Figure 2). Thus, a requirement about variability modeling is:

DA.Req1. *Support for complex variability modeling including clonable features, variable features, features with attributes, and complex constraints.*

WeaFQAs also considers NFPs such as performance and energy efficiency in the context of an FQA configuration. Although dealing with NFPs in feature models as attributes [10] is a wide accepted approach [27], it presents some issues. Firstly, NFPs usually compete and conflict with each other. For example, using the AES encryption algorithm has a high energy efficiency but provides a poor performance in execution time [34]. These relationships cannot be handled with the basic constraints of feature models (e.g., includes and/or excludes) and require more specific treatments (e.g., use of the NFR+ Framework) [18]. Secondly, sometimes the management of NFPs needs to be postponed until the requirements analysis or even product derivation phase because the NFPs' information about a feature is not available until the product is completely generated and tested, unless domain experts provide predictions and estimations of them. For example, to obtain the real energy efficiency or memory footprint of a product is necessary to evaluate it as a whole configuration and not as an independent features.

DA.Req2. *Support for dealing with and managing NFPs that can be associated to individual features or complete configurations.*

These two requirements appear in other current domains, such as the IoT [35, 55], where clonables features are essential to model the variability of the different devices, and NFPs need to be modeled when deployments have to be generated according to the tradeoff between different NFPs, such as latency and battery consumption.

Another challenge that WeaFQAs poses is the size of its variability model. Considering 20 FQAs modeled with a total of 178 features and 23 constraints, there are $5.72e24$ configurations. Variability models of this size are unmanageable and thus is desirable to modularize them, as for example, using *composite variability models* as defined for the CVL language [22] (see `Context` feature in Figure 2). Another solution to modularize an SPL that could fit very well with WeaFQAs is the use of a *multi product line* (MultiPLs) approach [58]. In a MultiPL approach, the FQAs for each quality attribute (e.g., the encryption, authentication and hashing FQAs for the security attribute) would be defined in separated SPLs. These SPLs would then be composed when used in a specific application.

DA.Req3. *Support for modeling/managing huge feature models.*

Requirements Analysis (RA). WeaFQAs allows creating the FQAs configurations according to the application requirements, but also generating optimum configurations based on NFPs, such as performance or energy efficiency. Generating optimum configurations is an intractable problem when dealing with huge variability models or models containing variable features. As discussed in

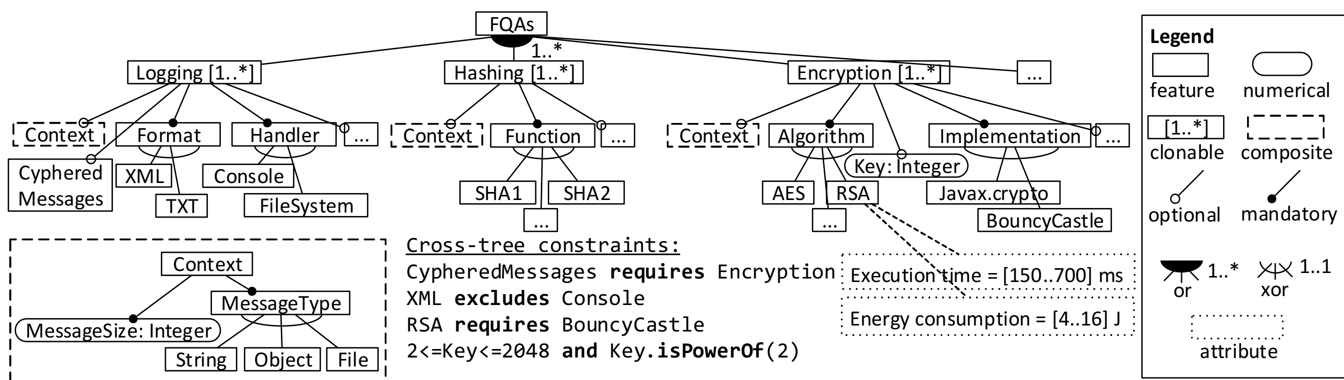

Figure 2: Excerpt of the WeaFQAs' variability model (complete model available in [36]).

`DA.Req2`, sometimes it is necessary to generate several configurations (or even all) in order to evaluate them and finding the best configuration before delivering the final product, or to reason about the system's variability. In those cases, specific formalizations of variability models (e.g., CNF [10] or BDD [25]) and reasoners and solvers (e.g., SAT [67], CPS [54]) need to be used depending on the type of analysis (e.g., calculate number of configurations, finding the optimum configuration, or generate all configurations).

RA.Req1. *Support for analyzing and generating optimum configurations from huge space configurations, based on different criteria (e.g., NFPs).*

In this phase, another activity that WeaFQAs considers is assisting application engineers when choosing a configuration. In this sense, WeaFQAs provides advises about the most appropriate selections based on the application engineers' goals. For example, to decide the most secure encryption algorithm or the most efficient framework that implements the RSA algorithm. This kind of assistant requires to manage partial configurations that will be completed in a step-by-step process.

RA.Req2. *Support for partial and step-by-step configurations.*

Domain Implementation (DI). There are frameworks and third party libraries that provide implementations of FQAs ready to be used, such as the Java Security package, the Apache Commons library, and the Spring Framework. So, implementing the FQAs' artifacts from scratch using a specific variability mechanism such as FOP or AOP is not an advisable option. Instead, the challenge posed by WeaFQAs is to handle the variability of existing artifacts in order to configure their functionality according to the selected features during the RA phase. Furthermore, the recurrent nature of the FQAs makes them suitable to be used in many different domains. Domains like web engineering involve multiple types of programming languages (e.g., JavaScript, Python, Groovy), and markup languages (e.g., HTML, CSS, XML) where applying typical variability development paradigms of SPL such as FOP or AOP is extremely difficult or even impossible [30].

DI.Req1. *Support for using and combining different variability mechanisms (FOP, AOP, annotations,...) independently from the language, and applying them to existing implementations.*

Additionally, WeaFQAs supports the management of FQAs at the architectural level. This means that WeaFQAs can handle architectural configurations of FQAs defined in any modeling language based on MOF (Meta-Object Facility), by using Model-Driven Engineering.

DI.Req2. *Support for managing artifacts variability at different abstraction levels: from software architecture models to code.*

Product Derivation (PD). In this phase, beyond the automatic generation of the product (here FQAs configurations), WeaFQAs has a specific need: the *weaving* of the FQAs into the final application. Following the WeaFQAs approach, where FQAs are modeled separately from the application, each generated configuration needs to be incorporated (woven) in different places of the application. In WeaFQAs this weaving activity can be performed together with the product derivation or in an independent activity (e.g., following a MultiPL [58]). In both cases, WeaFQAs supports the weaving process at the architectural level (using MOF-compliant models) or at the code level (by defining custom transformations) [29].

PD.Req1. *Support for weaving products or multi product lines.*

Finally, an important activity in an SPL process is the evolution engineering. When the applications' requirements change and/or the technology of the FQAs evolves, domain knowledge and artifacts need to be updated and the changes need to be propagated to the different deployed configurations. In some domains, as for example multi-tenants applications [32], with hundreds of configurations deployed, this activity should be performed automatically.

PD.Req2. *Support for automatic propagation of evolving changes to the deployed products.*

In the rest of the paper we take into account these requirements in order to answer our research questions.

4 STATE OF PRACTICE

This section answers our first research question:

RQ1: *What are the tools that provide some kind of support for SPLs? Are they available, usable, and keep up to date?*

We analyze the current state of practice of SPL tools to identify which ones are available online, and are really usable for industry,

Figure 3: State of practice of SPL tools.

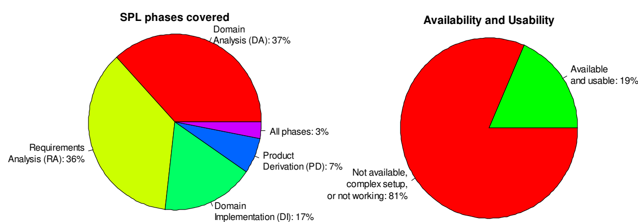

Figure 4: Summary of existing tools.

practitioners, and the SPL community. The goal is to collect all possible tools related with SPL to check their status before considering them to analysis. This does not pretend to be a systematic review of tools, but an in-depth study to identify existing tools.

Research method. We performed a manual search on different sources. First, we identified SLRs [6, 51] and surveys [43] about SPL tools. We also searched the proceedings of the demonstration and tool track in some of the most relevant events about SPL and variability (e.g., SPLC, VaMoS), for the period not covered by the SLRs and surveys (2015-2018). Second, 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 glanced at the tool by downloading, installing, and launching it to check its correct functioning.

Data extraction form. We used Google Forms[2] 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. This data has been extracted from the information found in reference papers, the official website, and the code repository of the tool. The only inclusion criteria for this form is that the tool is directly related with SPL or is being used in the context of SPL.

Result. To illustrate the state of practice, we have built a timeline (Figure 3) with all SPL tools published until February 2019[3]. We found 97 tools. As summarized in Figure 4, only 3% of them cover all phases of the SPL process. Moreover, there seem to be more interest in the problem space than in the solution space since the DA (37% of the tools) and the RA (36%) are the phases most covered by the tools. The DI and PD phases are only covered by 17% and 7% of the tools respectively. The main problem with SPL tools is the fact that only 19% are available online to be downloaded and used, have an easy installation process or can be directly used online in a web browser (green timelines in Figure 3). Characteristics that make them attractive to be used in the community and industry. The other 81% are obsolete tools, have complex installation process, work with errors, or they are not available at all (red timelines in Figure 3). This evidences that there are lot of tools but most of them are academic or prototypes tools that are abandoned shortly when the associated research project ends.

The state of practice gives a wide vision of the current state of art of the SPL tools and helps practitioners to select appropriate tools. The main artifacts developed that allows replicating and/or improving this state of practice are available in [36].

5 TOOL SUPPORT FOR COMPLEX SPLS

This section answers our second research question, selecting first a subset of the tools identified in Section 4:

RQ2: *How the tools behave when they are used in complex SPLs?*

5.1 Tool selection

Four inclusion criteria (IC) are considered relevant to answer RQ2:

IC1: The tool is available online to be downloaded or used.
IC2: The installation process is straightforward and does not require complicate settings, additional dependencies or third-party plugins that can be obsolete.
IC3: The tool is under active development or the latest release is a final stable version.
IC4: The tool covers at least one of the main phases of the SPL process defined in Section 3 and in [4, 52].

Five exclusion criteria (EC) were used to exclude tools that we do not consider appropriate to be used in a professional environment:

EC1: The tool is a prototype or a preliminary or beta version.
EC2: The tool does not work or works with errors.
EC3: The tool is commercial and the owner does not provide an alternative free limited version.
EC4: The tool is a compilation of independent tools not related to SPL (e.g., a CASE tool).
EC5: There is another similar, more actual or useful tool, or the tool has been integrated within another.

By applying our inclusion and exclusion criteria we have chosen the SPL tools to be analyzed in this section (Table 1). Note that there are many other tools that are available and usable (e.g., FeatureHouse [5], AHEAD [9],...), but they have been excluded by EC5 since they are integrated within other tools like FeatureIDE [64]. Others are very updated (e.g., ProductlineRE [26]) but have so many obstacles to be installed, so they do not pass IC2. Others are exclusive for a specific domain (e.g., FMCAT [7] focusing on the analysis of dynamic services product line). Others are generic tools that are not specific of SPL, even if they are used as part of some SPL tools in concrete phases of the SPL engineering process (e.g., AspectJ to implement artifacts following an AOP approach).

5.2 Tool analysis

In this section we analyze the selected tools to check whether or not they satisfy the requirements of a complex SPL process like the WeaFQAs process presented in Section 3. All artifacts developed and used throughout the different phases to test the tools are available to repeat the experiments in [36]. This includes the variability models in different formats and the implementation code of the artifacts.

5.2.1 ***Domain Analysis (DA) phase.*** Apart from variability and dependency modeling, the DA process should take into account requirements DA.Req1, DA.Req2, and DA.Req3.

Experiments. We have modeled the variability of the FQAs (Figure 2) with the selected tools. The WeaFQAs variability model includes clonable features (`Encryption [1..*]`), variable features (`MessageSize: Integer`), features with attributes (`Execution time` and `Energy consumption` of the encryption algorithms), composite units to modularize the variability model and to facilitate its management and configuration (`Context`), arbitrary group multiplicity (`1..*`), and complex cross-tree constraints like those involving variable features.

Analysis. Regarding DA.Req1., all tools support basic feature models (mandatory and optional features, alternative (xor) and or groups, requires and excludes constraints). However the support for advanced characteristic is very limited (Table 2). Clonable features is the most difficult characteristic to be implemented, and thus, no tool provides support for them completely. Clafer allows cloning any feature in the variability model and configuring each instance, but this is done at the configuration step and deciding whether a feature is clonable or not should be done at the domain analysis phase. FeatureIDE and pure::variants allow a similar behaviour of clonable features by inserting subtrees in the variability model. In FeatureIDE, this characteristic follows the VELVET approach of MultiPLs [58], but it is still a prototype that only supports the FeatureHouse [5] composition approach when generating code, it contains many errors and with not enough documentation. Pure::variants introduces the concept of "variant instance" as a link in the feature model to another configuration space. In contrast to

[2] https://forms.gle/JfH9bKHHTgCLc31R7
[3] A .csv file with the tools information is available in [36].

Table 1: Description of the selected SPL tools.

Tool	Year	Last update	SPL phases	Description
S.P.L.O.T. [45]	2009	Jan. 2015	DA, RA	Online tool to edit, debug, analyze, configure, share and download feature models. It offers hundreds of feature models from academics and practitioners. **Available in:** *http://www.splot-research.org/*
Glencoe [2]	2018	Jan. 2019	DA	Web application to work with variability models. Model importation from DIMACS or pure::variants [61]. Several solvers for the automated analysis of FMs. **Available in:** *https://glencoe.hochschule-trier.de/*
Clafer [3]	2014	Feb. 2018	DA, RA	General-purpose lightweight language for structural modeling: feature modeling and configuration, class and object modeling, and metamodeling. Several solvers and model reasoners. **Available in:** *https://www.clafer.org/*
FeatureIDE [42, 64]	2004	Nov. 2018	DA, RA, DI, PD	Open-source Eclipse framework with plug-in based extension mechanism to integrate and test existing tools and SPL approaches [39] (FeatureHouse, AHEAD, AspectJ, Antenna, etc.). **Available in:** *http://www.featureide.com/*
pure::variants [61]	2003	Dec. 2018	DA, RA, DI, PD	Commercial solution that supports all phases of the SPL process. Many extensions that connect pure::variants with common systems and software engineering tools [12]. **Available in:** *https://www.pure-systems.com/*
vEXgine [33]	2017	Jan. 2018	PD	Customizable implementation of the CVL execution engine [69] that can be extended with custom transformation engines to support multiple variability approaches. **Available in:** *http://caosd.lcc.uma.es/vexgine/*

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.

The support for variable features and for feature with attributes is confused because of the thin difference between them. Clafer allows defining variable features with a specific type (e.g., integer) that behaves as a normal feature but allows providing a value during the configuration step, and also specifying constraints about the value of that feature. However, to support attributes in Clafer (as for example to specify a utility value for each feature) we have to rely in the Clafer Multi-Objective Optimizer (ClaferMOO [3]) that is a specific reasoner for attributed-feature models, or modeling the attributes as variable (numerical) features. This implies to define an additional variable feature (e.g., integer) for each normal feature in the variability model, and make sure those variable feature are selected in the final configuration. Contrary, pure::variants offers complete support for attributes but not for variable features that in this case can be implemented as attributes. FeatureIDE supports attributes only partially, because it requires selecting the composer "Extended Feature Modeling" and then, no other composer can be selected. Neither S.P.L.O.T. nor Glencoe support clonable features, variable features, and feature with attributes.

Finally, each tool provides additional characteristics for variability modeling. For instance, Glencoe and pure::variants allow mixing mandatory features within "or" groups. Glencoe, Clafer and pure::variants 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. Clafer (with constraints involving variable features) and pure::variants (with Prolog, and its own variant of OCL: pvSCL [61]) allow defining complex constraints.

Concerning DA.Req2., no tool provides explicit support for dealing with NFPs, relying on features with attributes to manage NFPs.

Respecting DA.Req3., first, huge feature models cannot be easily modularized within existing tools. Clafer allows defining multiple feature models as abstract classes but all of them in the same file. FeatureIDE, as discussed for clonable features, supports MultiPLs but it is in its infancy and the feature models itself cannot be divided in multiple files. In pure::variants, the support is better since it defines an Hierarchical Variant Composition to link a feature

Table 2: Tool support for the Problem Space: DA and RA.

	Requirements and Characteristics	S.P.L.O.T.	Glencoe	Clafer	FeatureIDE	pure::variants
Domain Analysis (DA)	**DA.Req1**					
	Basic FMs (optional, mandatory, or, xor, requires, excludes)	■	■	■	■	■
	Cardinality-based FMs (clonable features or features cloning/cardinalities)	□	□	⧄	⧄	⧄
	Variable features (variable features with type — i.e., integer, string,…)	□	□	■	□	⧄
	Extended FMs (features with attributes)	□	□	⧄	⧄	■
	Other extensions (complex constraints, arbitrary group multiplicity, abstract features,…)	⧄	⧄	■	⧄	⧄
	DA.Req2					
	Support for NFPs	□	□	□	□	□
	DA.Req3					
	Modularization of FMs (composition units, multiple variability models)	□	□	⧄	⧄	■
	Evolution of FMs (modification of features — e.g., change variability type, move feature,…)	□	⧄	■	⧄	■
Requirements Analysis (RA)	**RA.Req1**					
	Analysis of FMs (statistics, validation,…)	■	■	⧄	■	■
	Number of configurations (model counting, independently of the FM's size)	■	■	⧄	⧄	■
	Generation of configurations (enumerate all configurations)	⧄	□	■	■	⧄
	Optimization of configurations (e.g., based on NFPs)	□	□	⧄	□	□
	RA.Req2					
	Partial configurations	■	□	■	⧄	⧄
	Step-by-step configuration	■	□	⧄	□	□

■ Full support. ⧄ Partial support. □ No support.

Table 3: Tool support for the Solution Space: DI and PD.

	Requirements/Characteristic	FeatureIDE	pure::variants	vEXgine
Domain Implementation (DI)	**DI.Req1**			
	Different variability mechanism (FOP, AOP, annotations,...)	■	■	⧄
	Multi-language / Language independent (used in the same project)	⧄	■	■
	DI.Req2			
	Model abstraction level (architecture, design, code,...)	⧄	⧄	⧄
Product Derivation (PD)	**PD.Req1**			
	Product derivation (variability resolution)	■	■	■
	Weaving products	□	⧄	■
	Multi Product Lines	⧄	□	□
	PD.Req2			
	Evolution changes (automatic propagation of changes)	□	⧄	⧄

■ Full support. ⧄ Partial support. □ No support.

model inside another one. Second, modifications of the variability models once created can be complex in some tools like S.P.L.O.T. and Glencoe, where modifying a part of the feature model usually can be only achieved by removing that part and adding it again. Contrarily, Clafer and pure::variants allow even moving features from a branch to another in a straightforward way.

Findings. Table 2 summarizes the characteristics supported by the analyzed tools. S.P.L.O.T. and Glencoe are the most usable tools for the DA 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 [23] that is the most comprehensible and flexible by now (and the most used) [10]. 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 feature model, 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 [69] (e.g., MoSIS CVL Tool [63] and BVR Tool Bundle [65]). However, those tools do not meet our inclusion/exclusion criteria because they are not currently available or are not in a usable state.

5.2.2 ***Requirements Analysis (RA) phase.*** The goal of this process is to select a desired combination of FQAs according to the application requirements. This phase should also consider the optimization of the FQA configurations based on NFPs, as well as assisting application engineers when generating the configurations.

Experiments. By using the selected tools, we analyze the possible configurations of the FQA's variability model, generating different (or all possible) configurations, and finding optimum configurations based on the NFPs (e.g., performance and energy efficiency in WeaFQAs) when possible. The challenge in this step is dealing with huge variability models such as the FQAs feature model.

Analysis. Regarding RA.Req1., almost all tools provide some kind of support for analyzing the variability model. This means statistics and metrics about the variability model (core features, optional features, number of constraints,...), model validation (consistency, void feature model,...), and anomalies detection (dead features, false-optional features, redundancy constraints,...). Clafer can only validate the model syntactically since it is a text plain modeling language with a formal grammar, but no further analysis about variability is carried out by default.

Depending on the requested analysis, each tool uses an specific feature model formalization and/or solver to perform the analysis. For example, to calculate the number of configurations or variability degree of the feature model, S.P.L.O.T. uses a Binary Decision Diagrams (BDD) engine [25] for which counting the number of valid configurations is straightforward. Glencoe uses a Sentential Decision diagram (SDD) [50] engine that enables to determine 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. However, these tools calculate the number of configurations without taking into account variable features, which considerably increments the total number of configurations. The other tools (Clafer and FeatureIDE) require to generate all configurations in order to enumerate them, and thus, with these tools is not possible to calculate the number of configurations for huge models, like the FQAs feature model, in a reasonable time. For instance, using the Choco solver [54] 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 of years to generate all configurations. FeatureIDE, in addition, generates the associated code, so it requires more time.

Additionally, none of the selected tools provide support for finding optimum configurations in feature models. Only Clafer, with its ClaferMOO module, provides a multi-objective optimization mode, but this implies to use another kind of model not related with the Clafer's variability model. Even so, with Clafer and FeatureIDE, we could generate all configurations (for small models), evaluate them based on the NFPs (e.g., performance and energy efficiency) and then finding the optimum configurations by using an specific optimization tool [34]. Other techniques such as random sampling applied to SPL [49] and formalizations of the feature models such as CNF or the use of advanced SAT solvers [14, 67] will allow reasoning about these aspects or help to solve these issues with huge models, but this is out of scope of this paper.

With respect to RA.Req2., only S.P.L.O.T. and Clafer provide a reasonable good support to manage partial configurations and step-by-step configuration. S.P.L.O.T. provides validation and statistics of the partial configuration, but also auto-completion of the configuration with less features or the configuration with more features. This is done through an online step-by-step configuration assistant. Clafer allows generating 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 that partial configurations, but they do not incorporate a guided process like S.P.L.O.T. to assist the user.

Findings. No tool allows generating all configurations efficiently for huge variability models like the required in WeaFQAs. Tools are able to calculate that the complete FQAs' feature model has $5.72e24$ different configurations, but without taking into account variable features (e.g., numerical values). In fact, nowadays, with the existing tool support for SPL it is not possible to generate optimum configurations of products based on some criteria like NFPs [34, 49].

5.2.3 *Domain Implementation (DI) phase.*

This phase focuses on the implementation of the variable artifacts (e.g., models, code).

Experiments. We have implemented some of the FQAs using different variability mechanisms, concretely AOP, FOP, and annotations. We have reused third party libraries like the Java Security package for the Hashing FQA, the BouncyCastle library for Encryption, and the SLF4J API for Logging.

Analysis. Regarding DI.Req1., FeatureIDE is the tool that provides best support for different variability mechanisms. Concretely, it supports FOP using the FeatureHouse approach or AHEAD, AOP with AspectJ, and annotations with Antenna (Java comments), Colligens (C preprocessor) or Munge (Android), among others [44]. However, it is not possible to combine different approaches in different parts of the application (e.g., annotations and AHEAD) or to use different languages (e.g., Java and JavaScript). Actually, only the combination of FeatureHouse with Java and AspectJ is supported.

The pure::variants tool also provides good support for AOP (e.g., AspectJ and AspectC++), and annotations (e.g., for Java, JavaScript, C++) with its own variation points system, but not for FOP. The *family model* [61] of pure::variants allows describing the variable architecture/code and to connect it via appropriate rules to the feature model. Nevertheless, most of the advanced options of pure::variants are only available in the commercial version. In the case of vEXgine, it is possible to use and combine different variability mechanisms but the resolution of that variability needs to be delegated to an external engine [33].

Concerning DI.Req2., FeatureIDE and pure::variants offer very good support for implementing the variability at the code level as discussed in DI.Req1, while vEXgine needs specific extensions to work at code level [30]. At a high abstraction level (architecture and design), both pure::variants and vEXgine offer the best support. However, pure::variants requires the commercial version to manage high abstract models (e.g., UML), and vEXgine requires to define the appropriate model transformations despite it supports any MOF-compliant model [33]. FeatureIDE offers the possibility to combine FeatureHouse and UML, but actually, this integration is not completely operable.

Findings. With existing tools, it is very difficult to apply the variability mechanisms (e.g., AOP, FOP) to third party libraries like those that implement FQAs, and the solution is usually encapsulating the behaviour of those libraries in entities of the specific approach (e.g., aspects or features) or implementing the FQAs from scratch using a specific variability mechanism. Moreover, no tool supports an effective variability mechanism to be applied over several languages (Java, Python, JavaScript) in the same project.

5.2.4 *Product Derivation (PD) phase.*

The product derivation phase is in charge of generating the final product (configurations of FQAs) by resolving the variability of the artifacts (FQAs) according to the selection of features made in the RA phase. Then, those configurations of the FQAs needs to be incorporated into the final application by some combination mechanism (weaving, or MultiPL). Also, the propagation of changes in the final products when the requirements changes or the domain artifacts evolve needs to be considered in this phase.

Experiments. We have generated the final products by resolving the variability of the FQAs with different configurations based on the configuration models specified in the RA phase. When possible, we have incorporated those configurations to existing applications. For example, weaving the FQAs' configurations to an electronic voting (e-voting) application [31]. Then, we have evolved the FQAs' variability model and try to update the generated configurations.

Analysis. Variability resolution and product derivation is achieved by all 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 to build and integrate a custom composer within FeatureIDE.

Apart from resolving the variability, PD.Req1. cannot be completely satisfied. In fact, only vEXgine provides complete support for weaving FQAs by defining custom model transformations [29]. The flexibility of pure::variants allows integrating other tools like Git to partially support mixing variants [59]. FeatureIDE integrates the VELVET approach [58] for MultiPL, but this is a prototype and in this case the product derivation is not fully operable.

Regarding PD.Req2. the support for propagating changes in the variability model to the existing configurations exists but is limited. In pure::variants, the source code of the product variants can be evolved by using merge operations from Git [28, 59]. Also, with the help of specific model transformations and evolution algorithms [31, 32], vEXgine can evolve the deployed artifacts, but the effort of defining those transformations is considerable.

Findings. Apart from the basic activity in this phase existing tools have not paid attention to advanced characteristics (e.g., weaving, MultiPL, evolution). However, those characteristic could be incorporate in some tools thanks to their extension mechanisms such as the possibility to define new composers in FeatureIDE [39] or the custom engines and model transformations of vEXgine [33].

6 SPL TOOLS ROADMAP

This section answers our third research question:

RQ3: *Is it possible to carry out a complete SPL process with the existing tool support?*

To answer RQ3, based on the analysis in the previous section, we define some practical roadmaps to completely carry out an SPL process with the existing tool support (Figure 5). The answer for RQ3 is that existing tools do support the complete process of SPL, but with many limitations. As shown in Figure 5, Roadmap 1 with

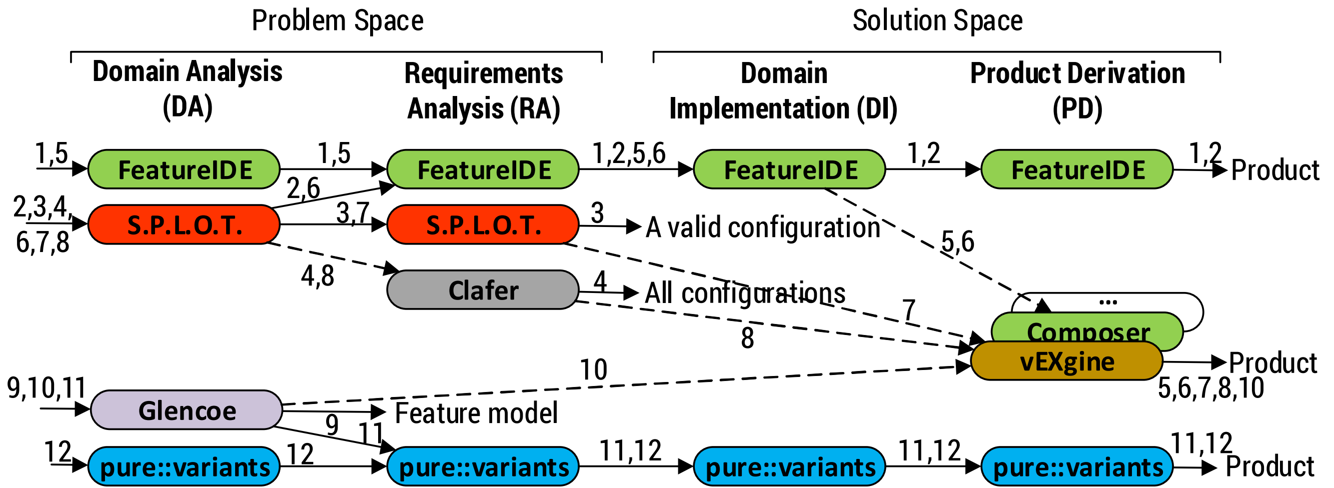

Figure 5: Roadmaps with the selected tools for an SPL process.

FeatureIDE and Roadmap 12 with pure::variants allows carrying out a complete SPL approach, covering the basic activities of an SPL process (variability modeling and artifacts implementation), and generating a final product. However, the limitations of these tools, as evidenced in Section 5, make them not suitable for complex SPL approaches like the WeaFQAs process that demands advanced SPL characteristics such as clonable features, managing huge models, or dealing with NFPs.

To partly solve these issues, SPL practitioners can combine some of the tools or integrate them. Following with our roadmaps (Figure 5), existing combinations are represented as solid lines, while possible combinations are represented as dashed lines. For instance, we can combine S.P.L.O.T. with FeatureIDE (Roadmaps 2 and 6) or Glencoe with pure::variants (Roadmap 11) to get the benefits of specifying the variability model in an online and easy to use web application like the offered by S.P.L.O.T. or Glencoe, and then loading the model in FeatureIDE or pure::variants, respectively. Since no standardized modeling format has been accepted after almost 30 years working with feature models, and the proposals for standardization (e.g., CVL [22], EMF [62]) have not jelled, each tool has defined its own format and notation, resulting in a high diversity of formats (e.g., SXFM, GUIDSL, Velvet, DIMACS,…). So, the roadmaps defined in this section will allow engineers and SPL practitioners to be aware about which tools can be used independently and in combination when a single tool does not support the complete SPL process.

In addition, when we are only interesting in analyzing the SPL variability, we can opt to use only Glencoe (Roadmap 9) that is the tool with best support for modeling and analyzing variability. When we need to generate an specific configuration (or a partial one) based on the requirements of the application, S.P.L.O.T. (Roadmap 3) offers an excellent feature-based interactive configuration module. When all configurations need to be generated at the RA phase, we can use Clafer (Roadmap 4). Note that we do not include a specific roadmap for Clafer because modeling the variability model in Clafer is a hard and tedious task that requires to learn a complex notation. Instead, to cover Roadmaps 4 and 8 we have developed a feature model converter from S.P.L.O.T. to Clafer. The implemented scripts and algorithms to fill some of the possible connections between the roadmaps are available in [36].

For implementing the variable artifacts from scratch (i.e., following a proactive and/or a reactive approach to develop an SPL [19]), FeatureIDE is the recommendable choice because it allows using several variability approaches (FOP, AOP, annotations) despite the fact that it does not allow directly combining those approaches (except for AOP the combination of which is straightforward). For those domains in which the applications require to combine more than one different approach (e.g., web engineering), practitioners will need to implement specific composers to allow the combination work, like a new composer plugin for FeatureIDE (Roadmaps 5 and 6). In this sense, vEXgine (Roadmaps 7, 8 and 10) provides great flexibility because it is design to be extensible by means of model transformations. For an extractive approach where practitioners start with a collection of existing products [19], pure::variants is a good choice thanks to its family model that connects the existing artifacts with the feature model.

Regarding NFPs, although there are specific tools to deal with NFPs such as the NFR+ Framework [18], these tools are not intended to be used in an SPL, and they have to be integrated within other SPL tools. The same occurs with those approaches that manage NFPs in an ad-hoc way by associating features and/or configurations to NFPs stored in a database [34, 46]. Actually, no tool provides even good support for modeling attributes in the feature model and manage them through the SPL process, as for example to generate optimum configurations based on these attributes.

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 [12]. Also vEXgine provides excellent support for resolving the variability of architectural models, but in this case the downside is that practitioners need to have some expertise in Model-Driven Engineering.

7 THREATS TO VALIDITY

This section discusses the threats to validity of this study [66]:

Internal validity. An internal validity concern is the reliability of the experiments to check the *functionality fulfillment* of tools. 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 interesting to carry out a complex SPL process, instead of reviewing all available functionalities provided by the tools.

External validity. An external validity concerns the *generalization/applicability of the results* to others SPL processes, beyond WeaFQAs. WeaFQAs was chosen because it follows the classical framework for SPL and incorporates additional requirements that can be found in current complex projects. Results of this work encompass from simpler SPLs [4, 52] to complex SPL processes like WeaFQAs or the Concern-Oriented Reuse (CORE) approach [1].

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

Missing important tools in the state of practice. 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), and in web search engines, and it was gathered through a data extraction form. We believe that we do not have omitted any relevant tool. However, since new tools are constantly appearing and evolving, we encourage practitioners to fill the information about any missed or new SPL tool 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 focuses specially on the availability and usability of the tools that we consider the first obstacle for their adoption in small/medium organizations. Therefore, we omit those tools that are not available to be directly downloaded, require to pay a licence, or with inadequate documentation because those tools are not capable of being analyzed before acquiring them.

Biased judgment selection and analysis. Due to the researchers involved in this study are active in the SPL research area and produced related tools (e.g., vEXgine, HADAS, Hydra, AO-ADL), a validity problem could be author bias. Only vEXgine passed our inclusion/exclusion criteria. 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 [33]; (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 Section 5.

Conclusion validity. Conclusion validity relates to the *reliability and robustness of our 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 completely or partially satisfy the exposed requirements. 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, experiments were carried out at least by two primary authors that acted as reviewers of the results reported by the other.

8 CONCLUSIONS AND FUTURE WORK

We have presented a state of practice of the tools for SPL, focusing on their availability and usability. Based on this study we have later empirically analyzed the most usable tools in order to check out the existence of enough mature tool support for carrying out a complete SPL process with demanding requirements. We have defined up to 12 different roadmaps 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 the activities/phases that are normally performed in complex SPLs. The main characteristics that tools should support are: (1) modeling variability of complex features (e.g., clonable features, variable features, composite features); (2) flexibility on the analysis of huge feature models considering 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 web engineering that could greatly benefit from the use of SPLs. Therefore, with the existing tool support is possible to carry out a simple SPL process but 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 incorporated to our roadmaps.

ACKNOWLEDGMENTS

This work is supported by the projects Magic P12-TIC1814, HADAS TIN2015-64841-R (co-financed by FEDER funds), MEDEA RTI2018-099213-B-I00 (co-financed by FEDER funds), and TASOVA MCIU-AEI TIN2017-90644-REDT.

REFERENCES

[1] Omar Alam, Jörg Kienzle, and Gunter Mussbacher. 2017. Modelling a family of systems for crisis management with concern-oriented reuse. *Softw. Pract. Exper.* 47, 7 (2017), 985–999. https://doi.org/10.1002/spe.2463

[2] Georg Rock Anna Schmitt, Christian Bettinger. 2018. 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. 665–673. https://doi.org/10.3233/978-1-61499-898-3-665

[3] Michał Antkiewicz, Kacper Bąk, Alexandr Murashkin, Rafael Olaechea, Jia Hui (Jimmy) Liang, and Krzysztof Czarnecki. 2013. Clafer Tools for Product Line Engineering. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops (SPLC'13 Workshops).* ACM, New York, NY, USA, 130–135. https://doi.org/10.1145/2499777.2499779

[4] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation.* Springer. https://doi.org/10.1007/978-3-642-37521-7

[5] S. Apel, C. Kästner, and C. Lengauer. 2013. Language-Independent and Automated Software Composition: The FeatureHouse Experience. *IEEE Transactions on Software Engineering* 39, 1 (Jan 2013), 63–79. https://doi.org/10.1109/TSE.2011.120

[6] Rabih Bashroush, Muhammad Garba, Rick Rabiser, Iris Groher, and Goetz Botterweck. 2017. CASE Tool Support for Variability Management in Software Product Lines. *ACM Comput. Surv.* 50, 1, Article 14 (March 2017), 45 pages. https://doi.org/10.1145/3034827

[7] Davide Basile, Felicita Di Giandomenico, and Stefania Gnesi. 2017. FMCAT: Supporting Dynamic Service-based Product Lines. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC '17)*. ACM, New York, NY, USA, 3–8. https://doi.org/10.1145/3109729.3109760

[8] Jonatas Ferreira Bastos, Paulo Anselmo da Mota Silveira Neto, Pádraig O'Leary, Eduardo Santana de Almeida, and Silvio Romero de Lemos Meira. 2017. Software product lines adoption in small organizations. *Journal of Systems and Software* 131 (2017), 112 – 128. https://doi.org/10.1016/j.jss.2017.05.052

[9] D. Batory, J. N. Sarvela, and A. Rauschmayer. 2004. Scaling step-wise refinement. *IEEE Transactions on Software Engineering* 30, 6 (June 2004), 355–371. https://doi.org/10.1109/TSE.2004.23

[10] David Benavides, Pablo Trinidad, and Antonio Ruiz Cortés. 2005. Automated Reasoning on Feature Models. In *Advanced Information Systems Engineering, 17th International Conference, CAiSE 2005, Porto, Portugal, June 13-17, 2005, Proceedings*. 491–503. https://doi.org/10.1007/11431855_34

[11] Thorsten Berger, Ralf Rublack, Divya Nair, Joanne M. Atlee, Martin Becker, Krzysztof Czarnecki, and Andrzej Wąsowski. 2013. A Survey of Variability Modeling in Industrial Practice. In *Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '13)*. ACM, New York, NY, USA, Article 7, 8 pages. https://doi.org/10.1145/2430502.2430513

[12] Danilo Beuche. 2016. Using Pure: Variants Across the Product Line Lifecycle. In *Proceedings of the 20th International Systems and Software Product Line Conference (SPLC '16)*. ACM, New York, NY, USA, 333–336. https://doi.org/10.1145/2934466.2962729

[13] Jan Bosch, Rafael Capilla, and Rich Hilliard. 2015. Trends in Systems and Software Variability. *IEEE Software* 32, 3 (2015), 44–51. https://doi.org/10.1109/MS.2015.74

[14] Agustina Buccella, Matias Pol'la, Esteban Ruiz de Galarreta, and Alejandra Cechich. 2018. Combining Automatic Variability Analysis Tools: An SPL Approach for Building a Framework for Composition. In *Computational Science and Its Applications – ICCSA 2018*, Osvaldo Gervasi, Beniamino Murgante, Sanjay Misra, Elena Stankova, Carmelo M. Torre, Ana Maria A.C. Rocha, David Taniar, Bernady O. Apduhan, Eufemia Tarantino, and Yeonseung Ryu (Eds.). Springer International Publishing, Cham, 435–451.

[15] Rafael Capilla, Jan Bosch, Pablo Trinidad, Antonio Ruiz Cortés, and Mike Hinchey. 2014. An overview of Dynamic Software Product Line architectures and techniques: Observations from research and industry. *Journal of Systems and Software* 91 (2014), 3–23. https://doi.org/10.1016/j.jss.2013.12.038

[16] Rafael Capilla, Alejandro Sánchez, and Juan C Dueñas. 2007. An analysis of variability modeling and management tools for product line development. In *Software and Service Variability Management Workshop-Concepts, Models, and Tools*. 32–47.

[17] Lianping Chen and Muhammad Ali Babar. 2011. A systematic review of evaluation of variability management approaches in software product lines. *Information and Software Technology* 53, 4 (2011), 344 – 362. https://doi.org/10.1016/j.infsof.2010.12.006 Special section: Software Engineering track of the 24th Annual Symposium on Applied Computing.

[18] Lawrence Chung, Brian A Nixon, Eric Yu, and John Mylopoulos. 2012. *Non-functional requirements in software engineering*. Vol. 5. Springer Science & Business Media.

[19] Paul C. Clements and Charles W. Krueger. 2002. Point - Counterpoint: Being Proactive Pays Off - Eliminating the Adoption. *IEEE Software* 19, 4 (2002), 28–31. https://doi.org/10.1109/MS.2002.1020283

[20] Kattiana Constantino, Juliana Alves Pereira, Juliana Padilha, Priscilla Vasconcelos, and Eduardo Figueiredo. 2016. 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 (ENASE 2016)*. SCITEPRESS - Science and Technology Publications, Lda, Portugal, 164–171. https://doi.org/10.5220/0005829801640171

[21] Maxime Cordy, Pierre-Yves Schobbens, Patrick Heymans, and Axel Legay. 2013. Beyond Boolean Product-line Model Checking: Dealing with Feature Attributes and Multi-features. In *Proceedings of the 2013 International Conference on Software Engineering (ICSE '13)*. IEEE Press, Piscataway, NJ, USA, 472–481. http://dl.acm.org/citation.cfm?id=2486788.2486851

[22] CVL Submission Team. 2012. Common variability language (CVL), OMG revised submission. http://www.omgwiki.org/variability/lib/exe/fetch.php?id=start&cache=cache&media=cvl- revised- submission.pdf..

[23] Krzysztof Czarnecki. 2002. Generative Programming: Methods, Techniques, and Applications Tutorial Abstract. In *Software Reuse: Methods, Techniques, and Tools*, Cristina Gacek (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 351–352.

[24] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. 2005. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice* 10, 1 (2005), 7–29. https://doi.org/10.1002/spip.213

[25] K. Czarnecki and A. Wasowski. 2007. Feature Diagrams and Logics: There and Back Again. In *11th International Software Product Line Conference (SPLC 2007)*. 23–34. https://doi.org/10.1109/SPLINE.2007.24

[26] Javad Ghofrani and Anna Lena Fehlhaber. 2018. ProductlinRE: Online Management Tool for Requirements Engineering of Software Product Lines. In *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 2 (SPLC '18)*. ACM, New York, NY, USA, 17–22. https://doi.org/10.1145/3236405.3236407

[27] F. Z. Hammani. 2014. Survey of Non-Functional Requirements modeling and verification of Software Product Lines. In *2014 IEEE Eighth International Conference on Research Challenges in Information Science (RCIS)*. 1–6. https://doi.org/10.1109/RCIS.2014.6861085

[28] Robert Hellebrand, Michael Schulze, and Uwe Ryssel. 2017. Reverse Engineering Challenges of the Feedback Scenario in Co-evolving Product Lines. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC '17)*. ACM, New York, NY, USA, 53–56. https://doi.org/10.1145/3109729.3109735

[29] Jose Miguel Horcas. 2018. *WeaFQAs: A Software Product Line Approach for Customizing and Weaving Efficient Functional Quality Attributes*. phdthesis. Universidad de Málaga. https://hdl.handle.net/10630/17231

[30] Jose Miguel Horcas, Alejandro Cortiñas, Lidia Fuentes, and Miguel R. Luaces. 2018. Integrating the common variability language with multilanguage annotations for web engineering. In *Proceeedings of the 22nd International Systems and Software Product Line Conference - Volume 1, SPLC 2018, Gothenburg, Sweden, September 10-14, 2018*. 196–207. https://doi.org/10.1145/3233027.3233049

[31] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2016. An automatic process for weaving functional quality attributes using a software product line approach. *Journal of Systems and Software* 112 (2016), 78–95. https://doi.org/10.1016/j.jss.2015.11.005

[32] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2016. Product Line Architecture for Automatic Evolution of Multi-Tenant Applications. In *20th IEEE International Enterprise Distributed Object Computing Conference, EDOC 2016, Vienna, Austria, September 5-9, 2016*. 1–10. https://doi.org/10.1109/EDOC.2016.7579384

[33] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2017. Extending the Common Variability Language (CVL) Engine: A Practical Tool. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC '17)*. ACM, New York, NY, USA, 32–37. https://doi.org/10.1145/3109729.3109749

[34] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2018. Variability models for generating efficient configurations of functional quality attributes. *Information & Software Technology* 95 (2018), 147–164. https://doi.org/10.1016/j.infsof.2017.10.018

[35] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2019. Context-aware energy-efficient applications for cyber-physical systems. *Ad Hoc Networks* 82 (2019), 15–30. https://doi.org/10.1016/j.adhoc.2018.08.004

[36] Jose Miguel Horcas, Mónica Pinto, and Lidia Fuentes. 2019. WeaFQAs' resources and artifacts. https://github.com/jmhorcas/SPLC2019.

[37] Claus Hunsen, Bo Zhang, Janet Siegmund, Christian Kästner, Olaf Leßenich, Martin Becker, and Sven Apel. 2016. Preprocessor-based variability in open-source and industrial software systems: An empirical study. *Empirical Software Engineering* 21, 2 (01 Apr 2016), 449–482. https://doi.org/10.1007/s10664-015-9360-1

[38] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier, and John Irwin. 1997. Aspect-oriented programming. In *ECOOP'97 — Object-Oriented Programming*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 220–242.

[39] Sebastian Krieter, Marcus Pinnecke, Jacob Krüger, Joshua Sprey, Christopher Sontag, Thomas Thüm, Thomas Leich, and Gunter Saake. 2017. FeatureIDE: Empowering Third-Party Developers. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B (SPLC '17)*. ACM, New York, NY, USA, 42–45. https://doi.org/10.1145/3109729.3109751

[40] Charles Krueger and Paul Clements. 2018. Feature-based Systems and Software Product Line Engineering with Gears from BigLever. In *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 2 (SPLC '18)*. ACM, New York, NY, USA, 1–4. https://doi.org/10.1145/3236405.3236409

[41] Miguel A. Laguna and Yania Crespo. 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Science of Computer Programming* 78, 8 (2013), 1010 – 1034. 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.

[42] Thomas Leich, Sven Apel, Laura Marnitz, and Gunter Saake. 2005. Tool Support for Feature-oriented Software Development: FeatureIDE: an Eclipse-based Approach. In *Proceedings of the 2005 OOPSLA Workshop on Eclipse Technology eXchange (eclipse '05)*. ACM, New York, NY, USA, 55–59. https://doi.org/10.1145/1117696.1117708

[43] Liana Barachisio Lisboa, Vinicius Cardoso Garcia, Daniel Lucrédio, Eduardo Santana de Almeida, Silvio Romero de Lemos Meira, and Renata Pontin de Mattos Fortes. 2010. A systematic review of domain analysis tools. *Information and Software Technology* 52, 1 (2010), 1 – 13. https://doi.org/10.1016/j.infsof.2009.05.001

[44] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, Thomas Leich, and Gunter Saake. 2017. *Mastering Software Variability with FeatureIDE*. Springer. https://doi.org/10.1007/978-3-319-61443-4

[45] Marcilio Mendonca, Moises Branco, and Donald Cowan. 2009. 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 (OOPSLA '09)*. ACM, New York, NY, USA, 761–762. https://doi.org/10.1145/1639950.1640002

[46] Daniel-Jesus Munoz, Mónica Pinto, and Lidia Fuentes. 2018. Finding correlations of features affecting energy consumption and performance of web servers using the HADAS eco-assistant. *Computing* 100, 11 (2018), 1155–1173. https://doi.org/10.1007/s00607-018-0632-7

[47] N Nazar and TMJ Rakotomahefa. 2016. Analysis of a Small Company for Software Product Line Adoption–An Industrial Case Study. *International Journal of Computer Theory and Engineering* 8, 4 (2016), 313.

[48] Lina Ochoa, Juliana Alves Pereira, Oscar González-Rojas, Harold Castro, and Gunter Saake. 2017. 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 '17)*. ACM, New York, NY, USA, 5–12. https://doi.org/10.1145/3023956.3023959

[49] Jeho Oh, Don Batory, Margaret Myers, and Norbert Siegmund. 2017. Finding Near-optimal Configurations in Product Lines by Random Sampling. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 61–71. https://doi.org/10.1145/3106237.3106273

[50] Umut Oztok and Adnan Darwiche. 2015. A Top-down Compiler for Sentential Decision Diagrams. In *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI'15)*. AAAI Press, 3141–3148. http://dl.acm.org/citation.cfm?id=2832581.2832687

[51] Juliana Alves Pereira, Kattiana Constantino, and Eduardo Figueiredo. 2014. A Systematic Literature Review of Software Product Line Management Tools. In *Software Reuse for Dynamic Systems in the Cloud and Beyond*, Ina Schaefer and Ioannis Stamelos (Eds.). Springer International Publishing, Cham, 73–89.

[52] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag, Berlin, Heidelberg.

[53] Christian Prehofer. 1997. Feature-oriented programming: A fresh look at objects. In *ECOOP'97 — Object-Oriented Programming*, Mehmet Akşit and Satoshi Matsuoka (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 419–443.

[54] Charles Prud'homme, Jean-Guillaume Fages, and Xavier Lorca. 2017. *Choco Documentation*. TASC - LS2N CNRS UMR 6241, COSLING S.A.S. http://www.choco-solver.org

[55] Clément Quinton, Daniel Romero, and Laurence Duchien. 2016. SALOON: a platform for selecting and configuring cloud environments. *Software: Practice and Experience* 46, 1 (2016), 55–78. https://doi.org/10.1002/spe.2311 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/spe.2311

[56] Mikko Raatikainen, Juha Tiihonen, and Tomi Männistö. 2019. Software product lines and variability modeling: A tertiary study. *Journal of Systems and Software* 149 (2019), 485 – 510. https://doi.org/10.1016/j.jss.2018.12.027

[57] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. 2018. A study and comparison of industrial vs. academic software product line research published at SPLC. In *Proceeedings of the 22nd International Conference on Systems and Software Product Line - SPLC '18*. 14–24. https://doi.org/10.1145/3233027.3233028

[58] Marko Rosenmüller, Norbert Siegmund, Thomas Thüm, and Gunter Saake. 2011. Multi-dimensional Variability Modeling. In *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems (VaMoS '11)*. ACM, New York, NY, USA, 11–20. https://doi.org/10.1145/1944892.1944894

[59] Sandro Schulze, Michael Schulze, Uwe Ryssel, and Christoph Seidl. 2016. Aligning Coevolving Artifacts Between Software Product Lines and Products. In *Proceedings of the Tenth International Workshop on Variability Modelling of Software-intensive Systems (VaMoS '16)*. ACM, New York, NY, USA, 9–16. https://doi.org/10.1145/2866614.2866616

[60] Gustavo Sousa, Walter Rudametkin, and Laurence Duchien. 2016. Extending Feature Models with Relative Cardinalities. In *Proceedings of the 20th International Systems and Software Product Line Conference (SPLC '16)*. ACM, New York, NY, USA, 79–88. https://doi.org/10.1145/2934466.2934475

[61] Olaf Spinczyk and Danilo Beuche. 2004. 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 (OOPSLA '04)*. ACM, New York, NY, USA, 18–19. https://doi.org/10.1145/1028664.1028675

[62] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. 2008. *EMF: eclipse modeling framework*. Pearson Education.

[63] Andreas Svendsen, Xiaorui Zhang, Franck Fleurey, Øystein Haugen, Gøran K. Olsen, and Birger Møller-Pedersen. 2010. CVL Tool - Modeling Variability in SPLs. In *Software Product Lines - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13-17, 2010. Workshop Proceedings (Volume 2 : Workshops, Industrial Track, Doctoral Symposium, Demonstrations and Tools)*. 299.

[64] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79 (2014), 70 – 85. 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).

[65] Anatoly Vasilevskiy, Øystein Haugen, Franck Chauvel, Martin Fagereng Johansen, and Daisuke Shimbara. 2015. The BVR Tool Bundle to Support Product Line Engineering. In *Proceedings of the 19th International Conference on Software Product Line (SPLC '15)*. ACM, New York, NY, USA, 380–384. https://doi.org/10.1145/2791060.2791094

[66] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, and Björn Regnell. 2012. *Experimentation in Software Engineering*. Springer. https://doi.org/10.1007/978-3-642-29044-2

[67] Yi Xiang, Yuren Zhou, Zibin Zheng, and Miqing Li. 2018. Configuring Software Product Lines by Combining Many-Objective Optimization and SAT Solvers. *ACM Trans. Softw. Eng. Methodol.* 26, 4, Article 14 (Feb. 2018), 46 pages. https://doi.org/10.1145/3176644

[68] Guoheng Zhang, Huilin Ye, and Yuqing Lin. 2014. Quality attribute modeling and quality aware product configuration in software product lines. *Software Quality Journal* 22, 3 (2014), 365–401. https://doi.org/10.1007/s11219-013-9197-z

[69] Ø. Haugen, B. Møller-Pedersen, J. Oldevik, G. K. Olsen, and A. Svendsen. 2008. Adding Standardized Variability to Domain Specific Languages. In *2008 12th International Software Product Line Conference*. 139–148. https://doi.org/10.1109/SPLC.2008.25