CASE Tool Support for Variability Management in Software Product Lines

RABIH BASHROUSH and MUHAMMAD GARBA, University of East London
RICK RABISER, CDL MEVSS, Johannes Kepler University Linz
IRIS GROHER, Johannes Kepler University Linz
GOETZ BOTTERWECK, Lero—University of Limerick

Software product lines (SPL) aim at reducing time-to-market and increasing software quality through extensive, planned reuse of artifacts. An essential activity in SPL is variability management, i.e., defining and managing commonality and variability among member products. Due to the large scale and complexity of today's software-intensive systems, variability management has become increasingly complex to conduct. Accordingly, tool support for variability management has been gathering increasing momentum over the last few years and can be considered a key success factor for developing and maintaining SPLs. While several studies have already been conducted on variability management, none of these analyzed the available tool support in detail. In this work, we report on a survey in which we analyzed 37 existing variability management tools identified using a systematic literature review to understand the tools' characteristics, maturity, and the challenges in the field. We conclude that while most studies on variability management tools provide a good motivation and description of the research context and challenges, they often lack empirical data to support their claims and findings. It was also found that quality attributes important for the practical use of tools such as usability, integration, scalability, and performance were out of scope for most studies.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Software and its engineering** → **Software product lines**; *Software notations and tools*; Software version control

Additional Key Words and Phrases: Software engineering, computer-aided software engineering, software variability

ACM Reference Format:
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.
DOI: http://dx.doi.org/10.1145/3034827

14

1. INTRODUCTION

Over the last two decades, Software Product Line (SPL) engineering has increasingly gained the attention of researchers and practitioners alike. This is due to the potential

The work of R. Rabiser was supported by the Christian Doppler Forschungsgesellschaft, Austria and Primetals Technologies. The work of G. Botterweck's was supported by Science Foundation Ireland (SFI) grants 10/CE/I1855 and 13/RC/2094.
Authors' addresses: R. Bashroush and M. Garba, School of Architecture Computing and Engineering, University of East London, United Kingdom; emails: r.bashroush@uel.ac.uk, garbamga@gmail.com; R. Rabiser, Christian Doppler Laboratory for Monitoring and Evolution of Very-Large-Scale Software Systems, Johannes Kepler University, Linz, Austria; email: rick.rabiser@jku.at; I. Groher, Department of Business Informatics—Software Engineering, Johannes Kepler University, Linz, Austria; email: iris.groher@jku.at; G. Botterweck, Lero—The Irish Software Research Centre, University of Limerick, Ireland; email: goetz.botterweck@lero.ie.

DOI: http://dx.doi.org/10.1145/3034827

economic advantages and business competitiveness the SPL engineering process can bring [Clements and Northrop 2007]. The benefits can range from cutting the development cost and increasing software quality to enabling mass customization, market dominance, and reduced time to market [Clements and Northrop 2007; Pohl et al. 2005a].

SPL engineering is about the planned reuse of common assets among a set of related systems, usually referred to as the *product line* or *product family* [Clements and Northrop 2007]. The SPL engineering process [Pohl et al. 2005a] involves studying and managing the common and varied features of the different product line members, a process usually referred to as domain engineering or development for reuse. Core (shared) assets—e.g., requirements, architecture, code, test cases—are then used as a basis to derive products from the product line, a process usually referred to as *application engineering* or *development with reuse*.

Defining and managing commonalities and variability in software product lines is widely referred to as variability management and is a key step of the SPL engineering process [van Gurp et al. 2001]. The variability management process guides the construction of product line variability models. Different types of variability models have been proposed, e.g., feature models, decision models, Orthogonal Variability Models (OVM), and UML-based approaches. In Section 1.1, we provide an overview of existing modeling approaches. For a detailed comparison and classification of variability modeling approaches, we refer to Czarnecki et al. [2012] and Sinnema and Deelstra [2007]. Variability models define the commonalities and variability of the product line from a problem space (e.g., features, decisions, or variation points) and a solution space (e.g., the reusable assets or variants) perspective along with the relationships that exist between these two spaces and among the elements in these spaces. Example relationships include exclusivity (when two features cannot exist in one product at the same time); inclusivity (when the existence of one feature depends on another); and alternatives (when only one of a number of alternative features can be supported), to name a few. Variability models tend to be very large in size, in many cases comprising thousands of features, and complex in nature due to the myriad of relationships that could exist among the features. This makes the construction of variability models manually a very tedious and error-prone process. Hence, tool support is of paramount importance for the variability management process.

Indeed, as widely acknowledged, not the least under the so-called triangle of success [Quatrani 2002], the success of software engineering projects depends on good tool support as much as on good software engineering processes. While existing work has investigated the variability management process in great detail—see, e.g., Chen and Babar [2011], for a systematic literature review of variability management approaches—tool support has not yet been studied in detail. In particular, existing work has not set out to identify and analyze all existing variability management tools. For the SPLE process, and particularly for variability management, a large number of tools have been developed over the last two decades. However, the majority of these tools had limited success with industrial adoption and never progressed beyond basic proof-of-concept stages. While dozens of experience reports on adopting the SPLE process in practice exist, very few focus specifically on variability management and even fewer on the tools supporting variability management [Berger et al. 2013].

In this work, we thus aimed to study all the published literature on CASE tool support for variability management over the last two decades using the systematic literature review methodology [Kitchenham et al. 2007]. The objective was to understand what tools have been produced, the characteristics of these tools, their context, and the challenges and limitations they faced. Particularly, given most of the academic tools never made it to industry, the aim was to document lessons learned and avoid

duplicate efforts in the future. This article presents the results of the study which: (i) will give practitioners access to a catalogue of published tools (commercial tools are discussed under a separate section) and guide them in selecting a tool for a given task enhancing the accessibility of published tools; (ii) provide researchers in the field with the main challenges and limitations that require further investigation; and (iii) provide new researchers with a good understanding of the state-of-the-art in tool support for variability management in SPL engineering.

The remainder of this section provides background overview of the various SPL modeling approaches. In Section 2, the research methodology is discussed. This includes the study's research questions, search protocol, inclusion and exclusion criteria, quality criteria, and the data extraction and synthesis process. Section 3 provides overall meta-analyses of the primary studies identifying trends in the field. Based on the data collected, the research questions are then addressed and discussed in detail in Section 4. Section 5 discusses additional findings on commercial tools and tool adoption in industry. Section 6 discusses the study limitations and threats to validity. Section 7 discusses related work. Finally, Section 7 rounds off with a summary and conclusions.

1.1. Background

Variability modeling is essential to define and document the commonalities and variabilities among a set of products in an SPL. Variability models are typically developed during domain engineering together with the reusable assets. During application engineering, products are derived by selecting variants in the variability model and reusing the corresponding assets developed during domain engineering [Pohl et al. 2005a].

Over the past two decades, a number of variability modeling techniques have been developed, each with its own characteristics and concepts [Sinnema and Deelstra 2007]. Feature modeling, decision modeling, and orthogonal variability modeling have gained the most attention[Czarnecki et al. 2012]. In this section, we provide a short overview of the three aforementioned techniques. Many of the CASE tools for variability management presented in this study support one of these three variability modeling techniques.

Feature modeling originates from the work on Feature-Oriented Domain Analysis (FODA) [Kang et al. 1990]. Feature models capture features, which are defined as end-user visible characteristics of systems in the domain Czarnecki et al. [2012]. Features are hierarchically organized in a feature tree and feature groups represent choices between multiple sub-features [Sinnema and Deelstra 2007]. Constraints and dependencies between features can typically be expressed in dedicated constraint languages. Feature models are used to model both the commonality and the variability of a set of products. Product derivation can be performed by selecting features from the feature model. Features can optionally be mapped to assets, which can be used to assemble products based on a valid feature selection during application engineering.

Decision modeling focuses on decisions that distinguish the products of a product line to guide product derivation. Czarnecki et al. [2012] define decisions as differences among systems, i.e., what needs to be decided on when configuring a system. Unlike feature modeling, which focuses on the documentation of commonality and variability, the main goal of decision modeling is to support product derivation during application engineering. Thus, a central concept in decision models is the mapping of decisions to reusable assets. Similar to feature models, dependencies and constraints between decisions can be defined, e.g., using constraint languages.

Orthogonal Variability Modeling [Pohl et al. 2005a] focuses on the documentation of variability in a separate variability model rather than integrating variability directly into development artifacts. Today, decision models and feature models are also primarily used in an orthogonal way [Czarnecki et al. 2012]. The variability defined in an OVM is related to other artifacts such as feature models, use case models, and component

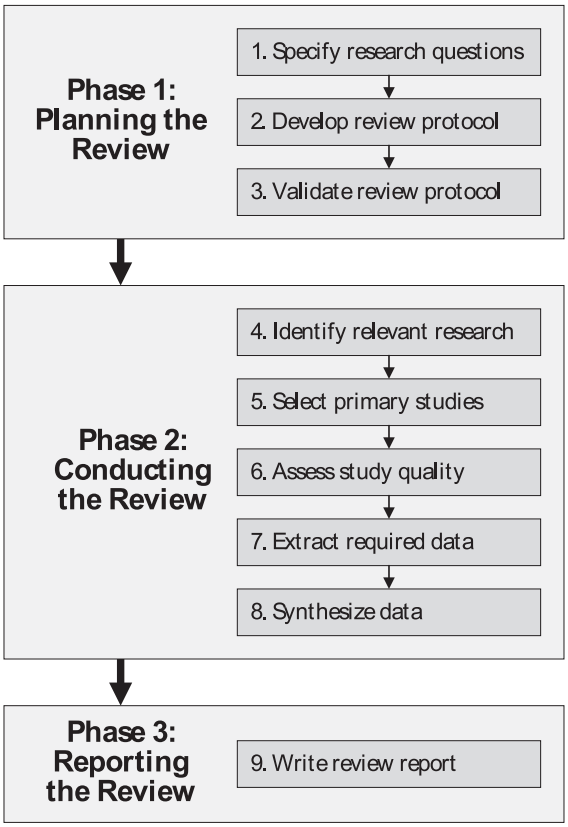


Fig. 1. Systematic Literature Review process [Brereton et al. 2007].

models [Pohl et al. 2005a]. OVM uses variation points and variants to capture the variability among a set of products. Similar to decision models, OVM focuses on documented variabilities. Constraint languages are used to define variability constraints.

In industrial settings, variability models in general not only tend to get very large but, due to their orthogonal nature, are linked to many different kinds of artifacts. Different viewpoints need to be provided to address different stakeholder concerns. This makes effective tool support for variability modeling inevitable before a practical application in industry can be performed. With various variability management tools developed over the years, this survey attempts to provide a systematic assessment of the state of the art.

2. RESEARCH METHOD

To achieve the objectives of this study, a Systematic Literature Review (SLR) approach was adopted. An SLR is a rigorous method for examining, evaluating, and interpreting all available research evidence based on research question(s) or particular research topic(s) [Kitchenham et al. 2007]. This study examines current literature on variability management tools in SPL engineering (known as primary studies) published over the last two decades. Throughout this research study, the guidelines for SLRs were followed as provided in Kitchenham et al. [2007]. This involves three main phases: (1) Planning the review; (2) Conducting the review, and; (3) Reporting the review. Figure 1 depicts the stages of SLRs, adapted from Brereton et al. [2007].

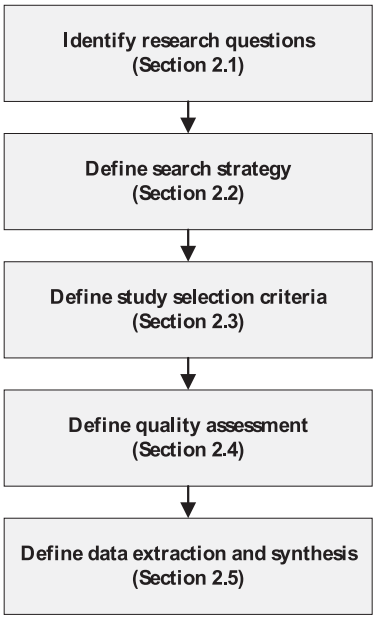


Fig. 2. SLR review protocol process [Brereton et al. 2007].

An important element in SLRs is the development of a review protocol (Figure 2). This protocol specifies the background and procedures to be used by researchers to ensure rigor while conducting the review and reduces the possibility of researchers' bias throughout the review process.

The systematic review protocol begins by defining research questions to be answered followed by the search strategy to be followed to identify the primary studies (described in Sections 2.1 and 2.2). Then, the study selection criteria for determining which studies should be included or excluded from the surveyed literature is defined (Section 2.3). Quality assessment criteria are then defined. These are used to assess the quality of the primary studies (Section 2.4). Finally, procedures for extracting and synthesizing data reported from primary studies are defined (Section 2.5).

Quality assessment and data extraction and synthesis have been performed by the authors of this article (four senior researchers, and one postdoc, with many years of experience in the area of variability management (the average experience is over 10 years)). For each article, the data were extracted by one of the four reviewers, who read the publication in detail. Another reviewer checked the extracted data. The articles were distributed randomly among the five reviewers, who did not extract or check their own publications.

2.1. Research Questions

In order to achieve the research aim and objectives of this study, we defined the following five research questions.

RQ1: What tools have been developed to manage variability in software product lines?
RQ2: What are the characteristics of these tools?
RQ3: What is the quality of the research conducted in the reported approaches?
RQ4: What is the context of research?

Table I. Electronic Databases Used for Searching for Primary Studies

S/No	Data Source Names
1.	IEEEXplore
2.	ACM Digital Library
3.	SpringerLink
4.	ScienceDirect
5.	CiteSeerXLibrary
6.	Microsoft Academic Search
7.	Scopus
8.	IEEE Computer Society Digital Library
9.	EBSCOhost E-Journal Services
10.	Google Scholar
11.	Web of Science

RQ5: What are the main challenges faced by current Product Line Management (PLM) tools?

2.2. Search Strategy

Following Kitchenham's guidelines [Kitchenham et al. 2007], we constructed a search string to help us identify the relevant primary studies to answer our five research questions.

≪Variability AND (Product Line OR Software Product Lines OR Software Product Family OR Software Product Families OR Product Family OR Product Families* OR Systems Family OR Family of Systems) AND (Variability OR Variability Management OR Variant OR Variation Point OR Feature Model OR Feature Modeling or Feature Modelling) AND (Tool OR Tools OR Approach, Approaches, Method* OR Methods)≫*

Although it was not possible to apply only one search string for all the electronic data sources, when varying the string for different sources we ensured that although the syntactic nature of the strings was not the same, they were all comparable semantically.

We also performed manual searches on different sources where SPL researchers were known to publish their findings; this included conferences and workshops. We searched for papers published between 1990 (i.e., when the first Feature-Oriented Domain Analysis (FODA) technical report was published [Kang et al. 1990]) up until July 2015 inclusive (when the search stage of this study was completed). Although only data reported in peer-reviewed published material was used in the analyses, we also attempted to acquire the identified tools. Where the tools were not available for download or use online, the respective authors were contacted.

Our search covered 11 digital data sources as shown in Table I. The manual search covered the proceedings of the following conferences and workshops:

SPLC (Software Product Line Conference)
VaMoS (Workshop on Variability Modelling of Software-Intensive Systems)
VisPLE (International Workshop on Visualization in Software Product Line Engineering)
WICSA (Working International Conference on Software Architecture)
EWSA (European Workshop on Software Architecture)

Finally, forward and backward reference checking ("snowballing") was conducted on the identified primary studies. Search engines were used to find citations of the primary studies identified that could be of relevance to the review (forward reference checking). The reference lists of the primary studies were then checked for any potentially relevant studies missed (backward reference checking).

2.3. Study Selection Criteria

This section explains the study selection process and lists the inclusion and exclusion criteria.

Inclusion Criteria (IC):

IC1: The primary study is a peer-reviewed, scientific paper rather than a PowerPoint presentation or a short/extended abstract paper.
IC2: The primary study discusses a variability management tool (i.e., a tool for which documenting and representing variability is a main aim and that was published in research associated with variability management, e.g., the SPL community).
IC3: When several reports of the same study existed in different sources, the most complete and recent version of the study was included in the review.
IC4: The paper was written in English.

Exclusion Criteria (EC):

EC1: The primary study does not address variability management tools.
EC2: The papers were published before January 1991 or after July 2015.
EC3: It is a short paper (<5 pages in two-column format, <8 pages in one-column format), PowerPoint file, poster presentation or consists of lecture notes.
EC4: The primary study consists of a compilation of work, for instance, from a conference or workshop.
EC5: The primary study discusses a commercial tool (we excluded commercial SPL variability management tools from this survey; we discuss the commercial tools separately in Section 5).

We found a total of 556 papers from different initial searches covering digital libraries, manual searches, and the works of known authors. After the initial screening of paper abstracts, in which papers addressing non-SPL related topics were excluded by one researcher, 113 publications were selected. The full papers were then acquired and four independent researchers reviewed the studies. Forty-seven publications were then selected through voting and discussions among the four researchers in a first step. Finally, and after another round carefully considering the inclusion and exclusion criteria, again through voting and discussions in case of disagreements, 37 studies were selected. Figure 3 shows a summary of the study selection process.

2.4. Quality Assessment Criteria

The quality of the reported research in the selected 37 papers was assessed based on the eight quality assessment questions listed in Table II. These were based on the quality assessment strategy defined in Kitchenham et al. [2007]. The studies were assessed using a ternary scale where each question was given a score of 1 (for Yes), 0.5 (for Perhaps), and 0 (for No). This system allowed us some flexibility when answering some of the questions that were difficult to judge as Yes or No from the information provided in the primary study. Once scores were allocated to questions, an aggregate mark was then given to each study. This data was also used to answer RQ3 (discussed in Section 3.4).

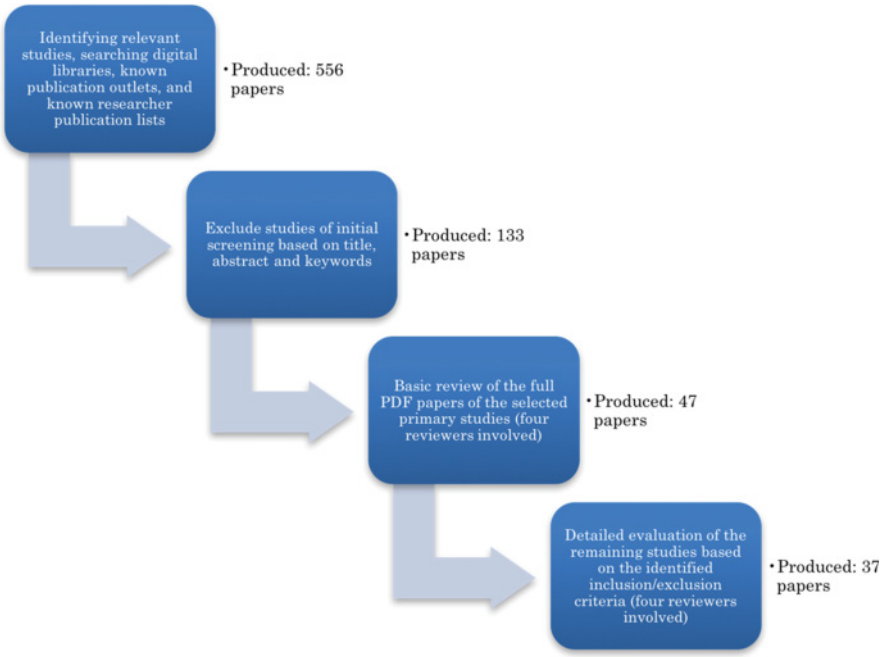


Fig. 3. Study selection process.

Table II. Quality Assessment Criteria

	Questions
QA.Q1	Is there a rationale for why the study was undertaken?
QA.Q2	Is there a description of the context (e.g., industry, laboratory setting, products used, etc.) in which the research was carried out?
QA.Q3	Did the paper present enough details about the variability management tool to enable us conduct the required analysis?
QA.Q4	Did the paper present an evaluation of the tool? If yes, did it include feedback from end users?
QA.Q5	Are the substantive claims in the paper supported by reliable evidence?
QA.Q6	Do the authors compare and evaluate their own results against related work?
QA.Q7	Do the authors discuss the credibility of their findings?
QA.Q8	Are limitations of the study discussed explicitly?

2.5. Data Extraction and Synthesis

Following the selection process, the 37 primary studies identified are shown in Table III.

Besides the 37 primary studies included in the study, we identified further 13 tools (in the 113 papers identified in the first round as described earlier) that did not meet the inclusion/exclusion requirements. These are shown in Table IV, along with the criteria they did not meet.

Upon the completion of the primary study selection phase, and the primary study quality assessment step, data extraction commenced. To answer the research questions, the following data was extracted from every primary study (cf. Table V). The following

Table III. Studies Included in the Final Review

Study ID	Paper Title	Year of Publication	Author(s)/ Reference
[S1]	DARE-COTS A Domain Analysis Support Tool	1997	[Frakes et al. 1997]
[S2]	Intelligent Design of Product Lines in Holmes	2001	[Succi et al. 2001]
[S3]	Scaling Step-Wise Refinement	2004	[Batory et al. 2004]
[S4]	XVCL: a mechanism for handling variants in software product lines	2004	[Zhang and Jarzabek 2004]
[S5]	Tool Support for Software Variability Management and Product Derivation in Software Product Lines	2004	[Gomaa and Shin 2004]
[S6]	XML-Based Feature Modeling	2004	[Cechticky et al. 2004]
[S7]	On the Implementation of a Tool for Feature Modeling with a Base Model Twist	2006	[Shakari and Møller-Pedersen 2006]
[S8]	COVAMOF: A Framework for Modeling Variability in Software Product Families	2004	[Sinnema et al. 2004]
[S9]	Towards Systematic Ensuring Well-Formedness of Software Product Lines	2009	[Heidenreich 2009]
[S10]	Odyssey: A Reuse Environment based on Domain Models	1999	[Braga et al. 1999]
[S11]	A NUI Based Multiple Perspective Variability Modeling CASE Tool	2010	[Bashroush 2010]
[S12]	The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study	2011	[Dhungana et al. 2011]
[S13]	XToF – A Tool for Tag-based Product Line Implementation	2010	[Gauthier et al. 2010]
[S14]	View Infinity: A Zoomable Interface for Feature-Oriented Software Development	2011	[Stengel et al. 2011]
[S15]	FeatureIDE: An Extensible Framework for Feature-Oriented Software Development	2012	[Thüm et al. 2012]
[S16]	FeaturePlugin: Feature Modeling Plug-In for Eclipse	2004	[Antkiewicz and Czarnecki 2004]
[S17]	An Integrated Software Management Tool for Adopting Software Product Lines	2012	[Park et al. 2012]
[S18]	Kumbang Configurator – A Configuration Tool for Software Product Families	2005	[Myllärniemi et al. 2004]
[S19]	Towards a Model-Driven Product Line for Web systems	2009	[Martinez et al. 2009]
[S20]	PuLSE-BEAT – A Decision Support Tool for Scoping Product Lines	2000	[Schmid and Schank 2000]
[S21]	Moskitt4SPL: Tool Support for Developing Self-Adaptive Systems	2012	[Gómez et al. 2012]
[S22]	BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models	2012	[Segura et al. 2012]
[S23]	An Analysis of Variability Modeling and Management Tools for Product Line Development	2007	[Capilla et al. 2012]
[S24]	Visualization of variability and configuration options	2012	[Pleuss and Botterweck 2012]
[S25]	ASADAL: A Tool System for Co-Development of Software and Test Environment based on Product Line Engineering	2006	[Kim et al. 2006]
[S26]	RequiLine: A Requirements Engineering Tool for Software Product Lines	2003	[von der Maßen and Lichter 2004]
[S27]	ToolDAy: A Tool for Domain Analysis	2011	[Lisboa et al. 2011]
[S28]	The Linux Kernel Configurator as a Feature Modeling Tool	2008	[Sincero and Schröder-Preikschat 2008]

(Continued)

Table III. Continued

Study ID	Paper Title	Year of Publication	Author(s)/ Reference
[S29]	Automating Product-Line Variant Selection for Mobile Devices	2007	[White et al. 2007]
[S30]	Managing Feature Models with FAMILIAR: a Demonstration of the Language and its Tool Support	2011	[Acher et al. 2011]
[S31]	WeEasy-Producer – Product Line Development for Variant-Rich Ecosystems	2014	[Eichelberger et al. 2014]
[S32]	OPTI-SELECT: an interactive tool for user-in-the-loop feature selection in software product lines	2014	[Yamany et al. 2014]
[S33]	MPLM - MaTeLo product line manager: [relating variability modelling and model-based testing]	2014	[Samih and Bogusch 2014]
[S34]	Variability code analysis using the VITAL tool	2014	[Zhang and Becker 2014]
[S35]	ViViD: a variability-based tool for synthesizing video sequences	2014	[Acher et al. 2014]
[S36]	VMC: recent advances and challenges ahead	2014	[ter Beek et al. 2012]
[S37]	WebFML: synthesizing feature models everywhere	2014	[Bécan et al. 2014]

Table IV. Studies Excluded in the Final Review

Reasons for Exclusion	Paper Title	Year of Publication	Author(s)/ Reference
EC3	FAMA Framework	2008	[Trinidad et al. 2008]
EC1	Development of a Feature Modeling Tool using Microsoft DSL Tools	2009	[Fernández et al. 2009]
EC3	S.P.L.O.T. - Software Product Lines Online Tools	2009	[Mendonca et al. 2009]
EC3	V-Manage	2008	[Sellier et al. 2008]
EC2	PACOGEN : Automatic Generation of Pairwise Test Configurations from Feature Models	2011	[Hervieu et al. 2011]
EC1	Variability Modeling in the Real: A Perspective from the Operating Systems Domain	2010	[Berger et al. 2010]
EC1	MetaProgramming Text Processor		[Campbell]
EC1	An Algorithm for Generating t-wise Covering Arrays from Large Feature Models	2012	[Johansen et al. 2012]
EC2&EC3	Varmod-Tool-Environment	2005	[Pohl et al. 2005b]
EC3	Linux Variability Analysis Tools (LVAT)		[She 2016]
EC2	VARMA–VARiability Modelling and Analysis Tool	2012	[Russell et al. 2012]
EC3	ZIPC SPLM	2009	[ZIPC Feature 2009]
EC3	Hydra Tool	2009	[Hydra Feature Modeling 2009]

data extraction form also shows the relevance of each of the extracted data elements to the study research questions.

3. DATA EXTRACTION AND META-ANALYSIS

The next step after the data extraction step was the data synthesis and analysis step. In this section, we provide meta-analyses of the primary studies relating to their publication types, venues, trends, and overall characteristics. In Section 4, we analyze the collected data to address the five main research questions of the study.

The first search of the systematic literature review resulted in 556 papers. The application of inclusion/exclusion criteria in several iterations resulted in 37 papers for the final review, which are listed in Table III.

Table V. Data Extraction Form

Data Field	Related Concern/ Research Question
DE.Q1 Paper title	Documentation
DE.Q2 Year of publication	Documentation
DE.Q3 Type of publication (e.g. Journal, Conference, Workshop, etc.)	Reliability of Review
DE.Q4 Publication outlet (conference name, etc.)	Reliability of Review
DE.Q5 Paper brief description (synopsis)	RQ1, RQ3
DE.Q6 The research rationale, challenges or problems as reported in the paper	RQ3, RQ5
DE.Q7 Research Context (e.g. industry, academic, product, etc.)	RQ4
DE.Q8 Tool Performance and Stability	RQ2, RQ5
DE.Q9 Visualization technique	RQ2
DE.Q10 Textual notation	RQ2
DE.Q11 Usability	RQ2
DE.Q12 Tool environment/Platform	RQ2
DE.Q13 Integration (e.g. with DOORS, etc.)	RQ2
DE.Q14 Scalability (ability to deal with large-scale models)	RQ2
DE.Q15 Relevance (Research or Practice)	RQ4
DE.Q16 The research limitations as reported in the paper	RQ5

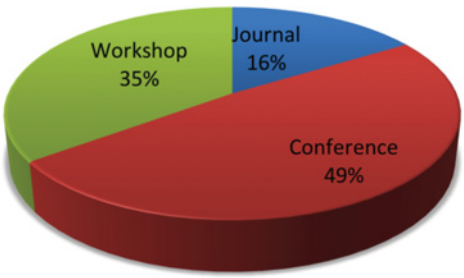


Fig. 4. Percentage of each publication type.

The primary studies included 18 conference papers, 6 journal papers, and 13 workshop papers. Figure 4 presents a pie chart showing the percentage for each publication outlet. From the chart, it can be seen that conferences are more prominent venues for research on variability management tools followed by workshops, whereas journals seem to be less attractive outlets for research on tools. The 37 papers are spread over 24 different venues. This distribution further highlights the importance of this systematic review as a manual search of well-known conferences or journals could not possibly identify all the relevant literature.

Figure 5 shows the distribution of studies over time. Our search did not identify any relevant paper published before 1997 and ended in 2014. The chart shows that there has been a considerable surge in new tools over the most recent years covered by our search.

4. DISCUSSION OF RESEARCH QUESTIONS

4.1. RQ1: What Tools have been Developed to Manage Variability in Software Product Lines?

Table VI provides a list of all the tools identified in the SLR in chronological order. A detailed analysis of the different tools is provided in the following sections.

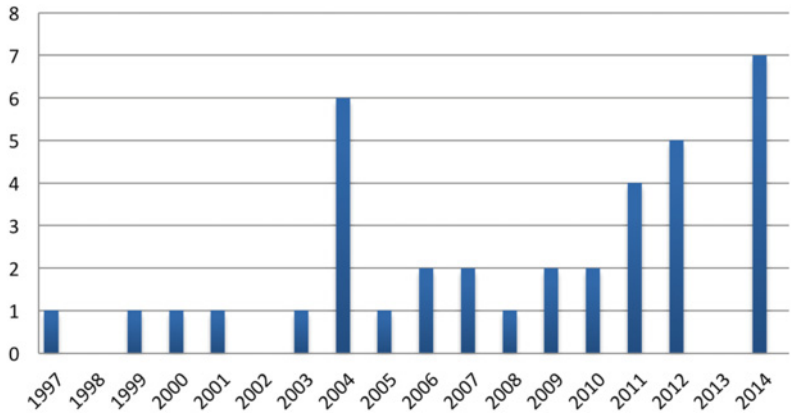


Fig. 5. Distribution of primary studies over time.

4.2. RQ2: What are the Characteristics of These Tools?

In this section, the tools identified are studied in terms of their development environment, support for transformations (between different formats), management of constraints and reasoning on variability models, and their proposed graphical and textual notations.

4.2.1. Development Environment. The described tools are based on different development environments. The most frequently named platform is Eclipse (16 studies), which includes tools based on the Generic Eclipse Modeling Framework, GEMS (1 study); Eclipse Rich Client Platform RCP application development (1 study); and the Eclipse Modeling Framework, EMF (9 studies). Within the latter group, two studies reported usage of textual modeling frameworks, i.e., EMFText [Heidenreich et al. 2009] and Xtext [Eysholdt and Behrens 2010], and three reported usage of graph-oriented UI frameworks, i.e., GMF [Eclipse 2016] and prefuse [Heer et al. 2005].

Two studies reported on tools based on commercial-off-the-shelf software, such as Microsoft Excel or Word. Six tools directly support the usage of UML, out of which two are based on commercial modeling tools, i.e., IBM Rational Rose and Rhapsody. Additionally, one study reported on a tool based on the C-preprocessor (CPP) code parser. Finally, three studies were web-based.

In terms of implementation languages, tools in 14 studies are based on Java, one tool is implemented in C# (RequiLine [S26]) and one in C (the Linux Kernel Configurator [S19]). The remaining 21 tools either do not state an implementation language or are realized as extensions of existing tools.

4.2.2. Transformation. Twelve studies reported the usage of some transformation mechanism, e.g., to support generating output. Two used XSL ([S6] and [S22]); one used dynamic loading of Simple XML Feature models (SXFM) [S32]; another used XML and Java source files [S31]; and one used the DIMACS format (a widely used standard for Boolean formulas in CNF) [S37].

4.2.3. Constraints and Reasoning. Fifteen studies reported on the usage of constraint languages or the usage of automated reasoning based on constraints in the wider sense. SAT solvers are used for instance by the S2T2 Configurator ([S24]), FAMILIAR [S30], and VMC [S36]; a CSP solver is for instance used by Scatter [S29] and [S35]; and propositional formulas by [S37].

Table VI. Identified Tools with Their Year of Introduction (Based on Publication)

Tool Name	Study ID	Year of Introduction
DARE-COT	[S1]	1997
Odyssey	[S10]	1999
PuLSE	[S20]	2000
Holmes	[S2]	2001
RequiLine	[S26]	2003
COVAMOF	[S8]	2004
Feature Modeling Plug-In	[S16]	2004
PLUSEE	[S5]	2004
XML-Based Feature Model	[S6]	2004
AHEAD	[S3]	2004
XVCL	[S4]	2004
KUMBANG	[S18]	2005
BVR: Base-Variation-Resolution	[S7]	2006
ASADAL (A System Analysis and Design Aid tooL)	[S25]	2006
Scatter Tool	[S29]	2007
VMWT	[S23]	2007
L K C- Feature Modeling Tool	[S28]	2008
FeatureMapper	[S9]	2009
PLUM	[S19]	2009
MUSA	[S11]	2010
XToF – A Tool for Tag-based Product Line Implementation	[S13]	2010
ToolDay	[S27]	2011
View Infinity	[S14]	2011
FAMILIAR	[S30]	2011
DOPLER	[S12]	2011
FeatureIDE	[S15]	2012
ISMT4SPL	[S17]	2012
BeTTy	[S22]	2012
MOSKitt4SPL	[S21]	2012
S2T2 Configurator	[S24]	2012
Easy-Producer	[S31]	2014
OPTI-SELECT	[S32]	2014
MPLM-MaTeLo product line manager	[S33]	2014
Variability code analysis using the VITAL tool	[S34]	2014
ViViD: a variability-based tool for synthesizing video sequences	[S35]	2014
VMC: recent advances and challenges ahead	[S36]	2014
WebFML: synthesizing feature models everywhere	[S37]	2014

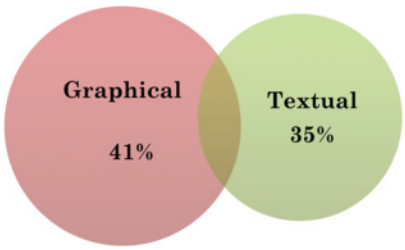


Fig. 6. Breakdown of tools based on the type of notation supported.

4.2.4. Graphical and Textual Notations. Among the 37 tools identified in the primary studies, some supported graphical notations only (15 tools), others textual notations only (13 tools), and few supported multiple notations and views (9 tools). Additionally, there were some that did not provide enough details on the notations supported. Figure 6

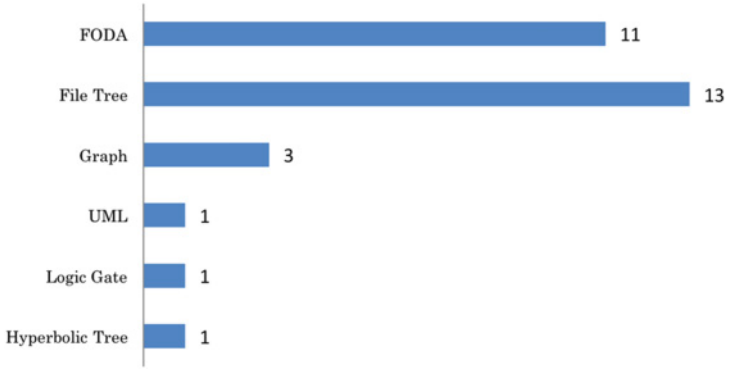


Fig. 7. Number of tools supporting each visualization type.

summarizes the breakdown of these notations based on the type of notation supported. These are discussed in details in the following subsections.

4.2.4.1. Graphical Notations. The graphical notations adopted by the tools reported in the primary studies can be classified under the following six visualizations:

FODA-like
File tree-like (vertical trees)
Graphs
Hyperbolic trees
Logic diagrams (logic gates)
UML

Figure 7 shows the number of tools supporting each visualization type. The figure shows that FODA-like and File tree-like representations are still the most popular approaches.

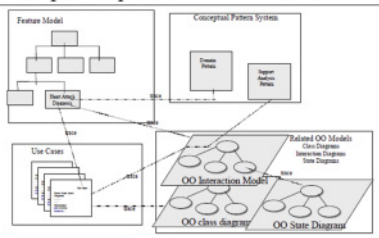
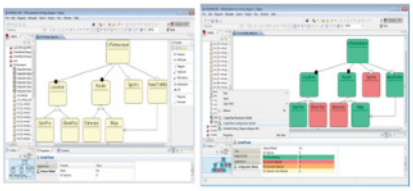
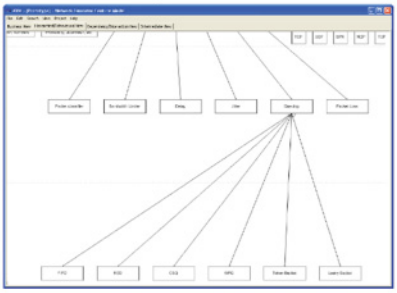
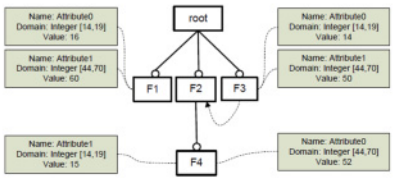
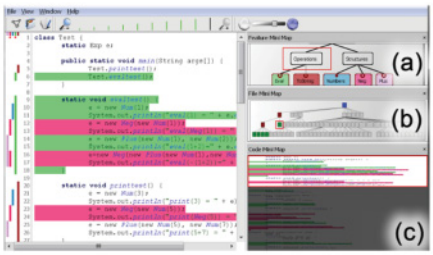
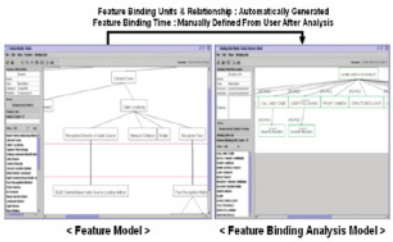
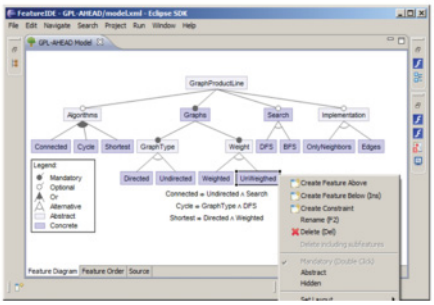
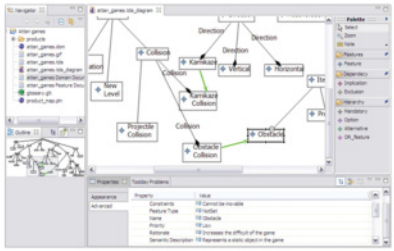
Tools in 11 studies are based on the FODA (Feature-Oriented Domain Analysis [Kang et al. 1990]) approach. These are:

[S10], FODA with UML
[S11], FODA, hyperbolic trees, logic diagrams and file tree
[S14], FODA, zoom-able interface to color-coded source code
[S15] and [S25], FODA with color coding
[S17], FODA multiple trees per feature model
[S21], FODA with color coding and basic file tree
[S22], FODA basic feature tree with attributes
[S27], FODA, UML and basic file tree
[S30], FODA, basic file tree and coding area
[S37], FODA and basic file tree

Examples of these notations are shown in Table VII (snapshots taken from the corresponding primary studies). As can be seen in the table, different tools use different parts of the interface to display the FODA-like feature model. As such, they are all prone to graphical overloading issues, where once the feature model size gets into the hundreds, it becomes cumbersome to browse and manage.

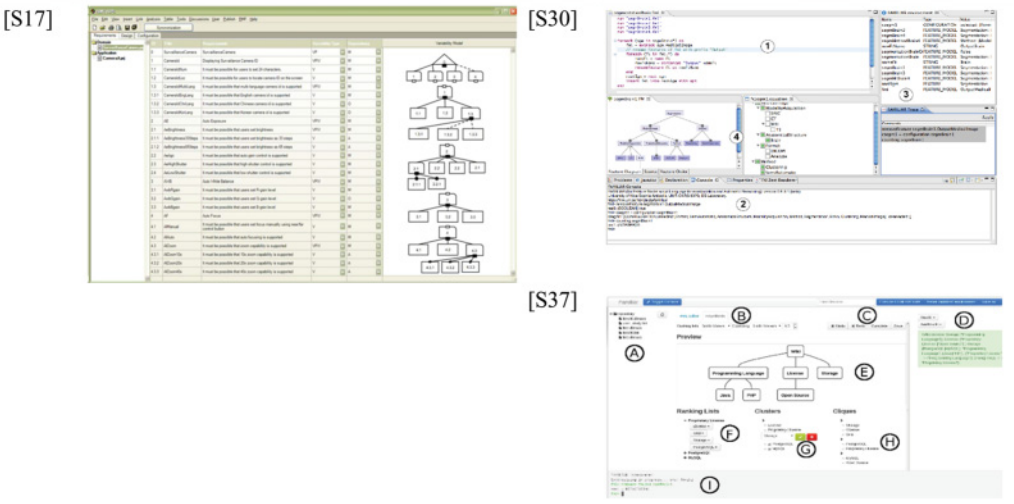
Thirteen tools adopt file tree approaches of which 8 used basic right click functionality to access information (tools reported in studies [S7], [S9], [S13], [S26], [S28], [S31], [S32], and [S33]). Two studies are based on advanced customization (color, shapes, etc.) of feature icons (tools in studies [S12] and [S16]). One study reports file trees with

Table VII. Tools with FODA-like Visual Notations

Study	Example Snapshot	Study	Example Snapshot
[S10]		[S21]	
[S11]		[S22]	
[S14]		[S22]	
[S15]		[S27]	

(Continued)

Table VII. Continued



semi-circles representing relationships among different features [S8]. Flow maps are also used in [S24].

A summary of these notations is shown in Table VIII. As can be seen in the table, this family of tools tends to be more scalable due to the inherent nature of the file tree navigation mechanism. However, they are not as good as FODA-like tools in enabling better intellectual control over the model (textual abstraction vs. graphical abstraction).

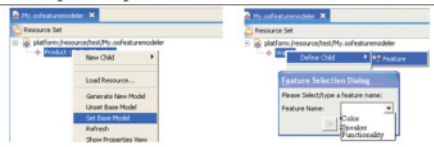
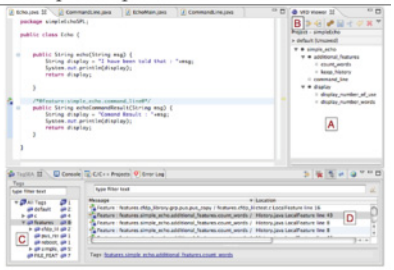
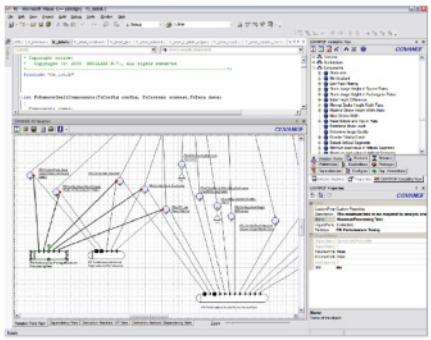
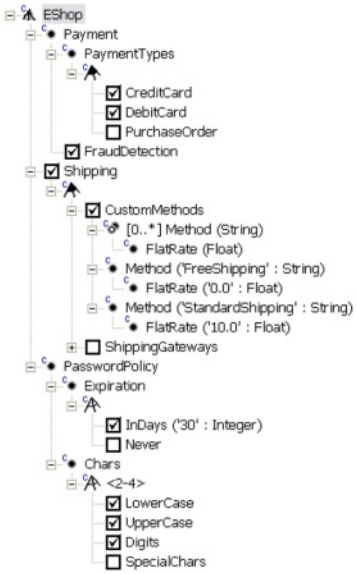
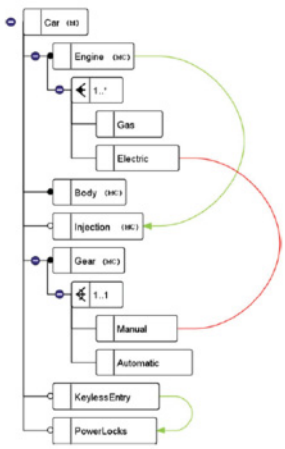
Three tools support graph-based visualizations, one includes a configuration interface using simple node-link graphs (user flows) with different objects [S2]; another tool supports the use of different objects for dependencies (circles, triangles, etc.), file tree, and coding area [S8]; and one tool is based on KOALA [van Ommering et al. 2000] like graph visualization, i.e., it is architecture-centric [S18]. Additionally, one tool adopts a logic diagram (schematics) visualization approach [S11]; another provides a UML-based visualization [S5]; and one adopts hyperbolic tree visualization [S11].

Examples of these visualizations are shown in Table IX. Looking at the table, it can be seen that notations that adopt hyperbolic views tend to have the best balance between scalability and intellectual control (abstraction). While managing to display the structure of the complete feature model, hyperbolic trees allow for browsing the model by displaying more details about nodes that are centered in the middle of the screen, allowing for smoother navigation capabilities, especially when paired with Natural User Interface (NUI) capabilities (e.g., pinching for zooming, etc.).

There are studies that do not provide enough details on the graphical notation used in the tools described ([S19] and [S29]).

Overall, seven tools supported multiple views of the feature model, where combinations of a graph, a file tree, and a coding area are used by [S8]; Koala and file tree is reported in [S18]; a file tree and a coding area are used in [S13] and [S31]; FODA and basic file trees are used in [S21] and [S37]; FODA, a basic file tree and a coding area are reported in [S30], FODA, UML and a basic file tree are used by [S27]; and FODA, hyperbolic trees, logic gates and a file tree are reported in [S11] as summarized in Figure 8.

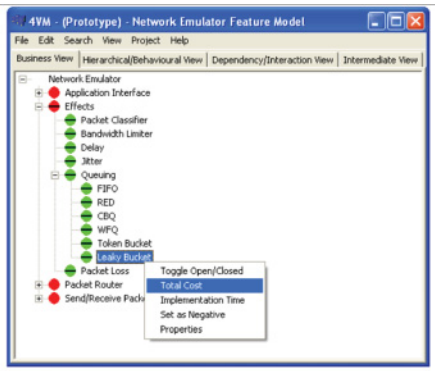
Table VIII. Tools with File-Tree-Like Visualizations

Study	Example Snapshot	Study	Example Snapshot
[S7]		[S13]	
[S8]		[S16]	
[S9]		[S24]	

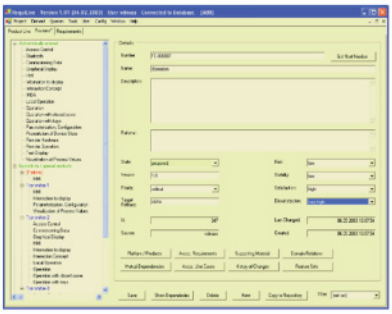
(Continued)

Table VIII. Continued

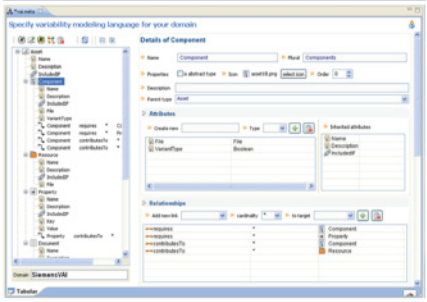
[S11]



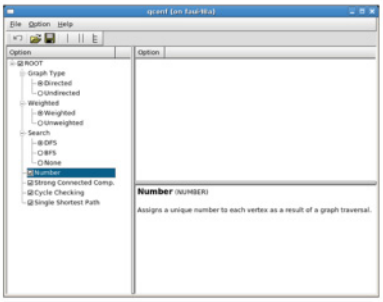
[S26]



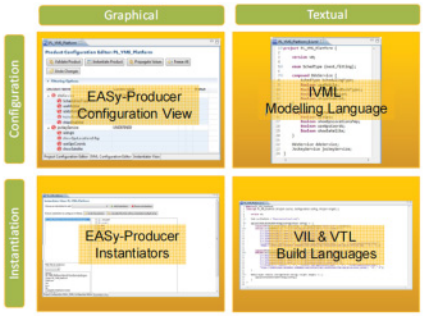
[S12]



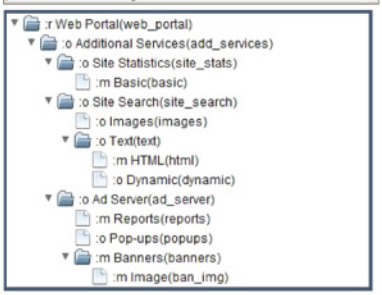
[S28]



[S31]



[S32]



[S33]

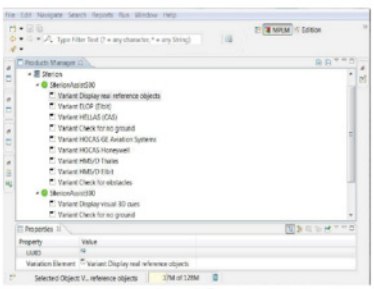
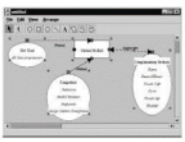
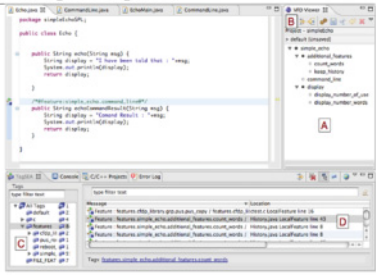
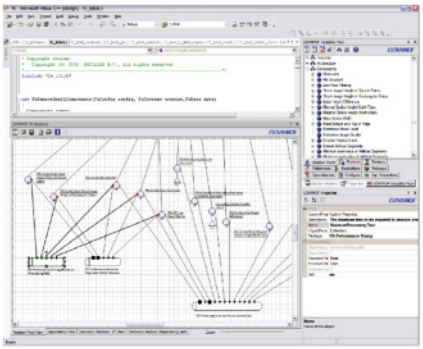
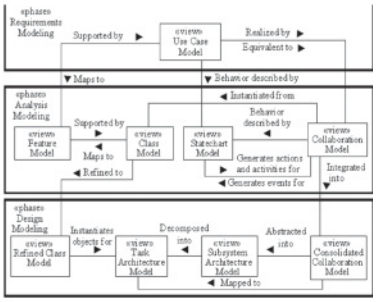
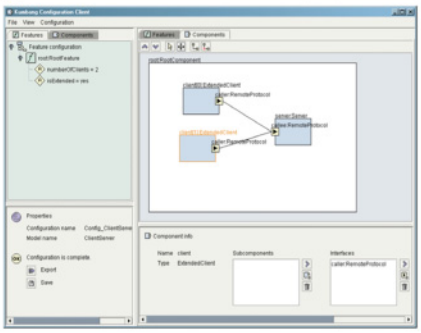
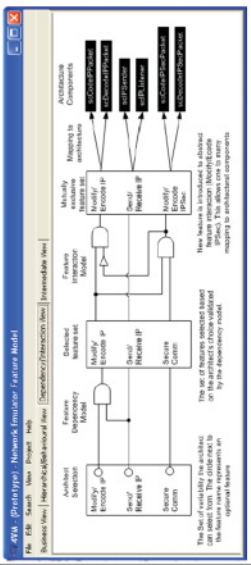


Table IX. Tools with Graph, Logic Diagrams, UML, and Hyperbolic Tree Visualizations

Study	Example Snapshot	Study	Example Snapshot
[S2] Graph	 	[S11] Hyperbolic Tree	
[S8] Graph		[S5] UML	
[S18] Graph		[S11] Logic Diagrams	

4.2.4.2. Textual Notations. For the textual notations, tools in 13 studies reported the use of textual notations. These can be classified under three different categories:

Code-like: with syntax similar to programming languages
XML-based: notations that are based on XML
Code-based: notations that embed variability representation within source code

Figure 9 shows the number of tools supporting each textual notation type.

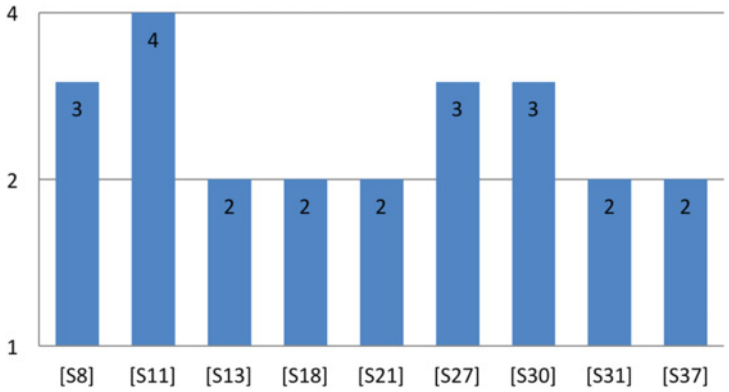


Fig. 8. Number of views per tool for tools with more than one view.

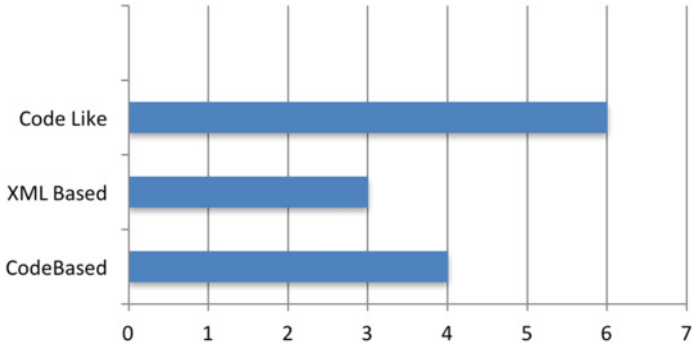


Fig. 9. Tools with various textual notations.

Code-like notations can be found in the tools described in [S3], [S18], [S28], [S30], [S34] and [S36]. Example snapshots of these notations can be found in Table X.

XML-based notations are supported in [S4], [S8], and [S22]. Samples of these notations are presented in Table XI.

Finally, Code-based notations are found in [S7], [S13], [S14], and [S35]. These are demonstrated in Table XII.

The preceding samples demonstrate that it can be difficult for humans to read all of these descriptions, whether they are written in code-like, code-based or XML format. When a textual notation is concise and easily readable and editable, many developers often prefer to work with simple text editors over a graphical user interface; However, for very complex notations (too complex to edit manually), it is essential to provide GUI-based tool support. With regard to variability management tools, there is a need to develop a standardized description format to allow better exchange of information among the different tools (such as the CVL initiative, see http://www.omgwiki.org/variability/). Competition between different tools would then be based on the quality of presentation and intuitiveness of navigation of such information by end-users, be it a textual or graphical notation.

Finally, there were six further notations that did not provide enough details in the paper about the textual notations they support, namely [S1], [S6], [S15], [S20], [S23], and [S24].

Table X. Tools with Code-like Textual Notations

[S3]

```
gui:                main:common
  compile G           compile A
                      compile B
lnk: gui main         compile C
  link gui main
                    common:
common:               compile X2
  compile X2
                  gui:
clean:                compile G
   delete *.gif
   super.clean    lnk: gui main
                    link gui main

                  clean:
                    delete *.gif
                    delete *.class
```

[S28]

```
config GPL
   boolean "ROOT"
   select M1


choice
   depends on GPL
   prompt "Graph Type"

   config DIRECTED
      boolean "Directed"

   config UNDIRECTED
      boolean "Undirected"
endchoice

config NUMBER
   default y if GPL
   requires (BFS || DFS)
   boolean "Number"
```

[S18]

```
Kumbang model KumbangExample
   root feature FSystem;
   root component CSystem

feature FSystem {
   subfeature
     (FeatureA, FeatureB) f;
   implementation
      instance_of(f, FeatureA)
<==> value($, attr) = a;
      instance_of(f, FeatureB)
<==> value($, attr) = b;
}

feature FeatureA {}
feature FeatureB {}

component CSystem {
   attributes
      ABBalue attr;
}

attribute type ABValue = 42
```

[S30]

```
GraphicCard:
DirectX Bus [Vertex];
// Vertex is optional

DirectX: (v10 | v10.1)+; // Or-group

Bus: (n64 | n128);
// Alternative-group

n64 -> Vertex;
// Constraints
```

[S34]

```
Variability Code Metrics
Supported in VITAL

Metric              Description

VP Nesting Degree   #ifde
                    nesting
                    level of a
                    given VP

Var Tangling Degree  #Vars used
                     in a given
                     VP
```

[S36]

```
Station(I,N,J,M) =
( [N = 0] nobike(I).Station(I,N,J,M) +
[N > 0] bike(I).Station(I,N-1,J,M) ) +
return(I).Station(I,N+1,J,M) +
redistribute(may,?FROM,?TO,?K).
( [TO = I] Station(I,N+K,J,M) +
[TO /= I] Station(I,N,J,M) ) +
[N > M] redistribute(may,I,J,N-
M).Station(I,M,J,M)
Users(I,J) =
request(I).
( bike(I).return(J).Users(I,J) +
nobike(I).Users(I,J) )
```

(Continued)

Table X. Continued

Var Fan-out on VPG	#VPGs that contain a given Var
Var Fan-out on File	#files that contain a given Var
Var Fan-in on File	#Vars included in a given File
VP Fan-in on File	#VP included in a given file

4.3. RQ3: What is the Quality of the Research Conducted in the Reported Approaches?

We analyzed the quality of research using the quality scores (0, 0.5, 1) for the eight quality questions (cf. Section 2.4) and also assessed how the studies address four different quality attributes important for tools usability, integration, scalability, and performance.

Table XIII presents the results of the quality assessment of the 37 studies included in the final review according to the quality questions. A frequency analysis of the scores for each quality question is presented in Figure 10. Most studies (92%) provide a rationale for why the study was undertaken (Q1). Almost half of the studies (41%) describe the context in which the research was carried out (Q2), 27% at least partially describe the context. More than half of the papers (60%) described the variability management tool in enough detail to be able to perform an in-depth analysis of the capabilities of the tool (Q3). Only one paper did not describe the tool at least partially. Very few studies (0.05%) present an evaluation of their proposed tools including feedback from end users (Q4). Over 60% of the studies do not evaluate their tool at all. Less than a third of the studies (30%) support substantive claims made in the paper with reliable evidence (Q5). Less than a third (30%) of the studies compare and evaluate their own results against related work (Q6). Finally, very few studies (16%) discuss the credibility of their findings (Q7) and limitations (Q8).

In general, the authors provided a motivation and a description of the research context, but papers lacked data to support the claims and findings. Also, authors seldom provided a critical reflection of their results. Even though the tools were described well in the papers, most variability management tools presented were not well evaluated, especially with respect to feedback from end users. The lack of industrial validation and evidence could be an important factor limiting the industrial adoption of these tools.

Table XIV presents different quality attributes we focused on in our review (usability, integration, scalability, and performance) and how well they were addressed by the studies. The quality attributes were identified through an interview-based survey conducted with a number of SPL practitioners who were asked to list their five most important attributes of an SPL tool. Figure 11 shows the frequency analysis of the results for each quality attribute. As can be seen, most studies do not mention the attributes explicitly with only a few studies providing contributions to the different areas of the quality attributes. Interestingly, none of the tools contributes and evaluates usability.

Table XI. Tools with XML based textual notations

Study	Example Snapshot	Study	Example Snapshot
[S4]	<x-frame name=„Being“ language=“java“> <set var=“BEING_CLASS“ value=“Being“/> <break name=“BEING_PARAMETERS“ /> class <value-of expr=“?@BEING_CLASS?“/> />{ String Name; int Age; double Weight; double Height; <break name=“BEING_BODY“/> public String getName(){return Name;} public int getAge(){return Age;} public double getWeight(){return Weight;} public double getHeight(){return Height;} <break name=“BEING_NEW_METHODS “/> }; </x-frame>	[S22]	GeneratorCharacteristics characteristics = new GeneratorCharacteristics(); //number of features characteristics.setNumberOfFeatures(30); //percentage of constraints characteristics.setPercentageCTC(10); //Max number of products of the feature model to be generated characteristics.setMaxProducts(1000); IGenerator generator = new MetamorphicFMGenerator(new FMGenerator()); FaMaFeatureModel fm = (FaMaFeatureModel)generator.generateFM(characteristics); System.out.println(“Number of products of the feature model generated: “ + generator.getNumberOfProducts()); FMWriter writer = new FMWriter(); writer.saveFM(fm, “./model.xml”); //FaMa XML format writer.saveFM(fm, “./model.afm”); //FaMa textual format
[S8]	<variationpoint id=”[id]”> <artefact> [artefact identifier] </artefact> <abstractionlayer> [abstraction layer] </abstractionlayer> <description> [description] </description> <type> optional | alternative | optional variant |variant | value <type> <variants> <!-- if not type=value --> <variant id=”[id]”> . . . <variant id=”[id]”> </variants> <range> [range specification] </range> <!-- if type=value -->		

(Continued)

Table XI. Continued

```
    <state>
      open | closed
    </state>
    <mechanism>
      [mechanism]
    </mechanism>
    <bindingtime>
      [bindingtime]
    </bindingtime>
    <rationale>
      [rationale]
    </rationale>
  </variationpoint>
```

Table XII. Tools with Code-Based Textual Notations

[S7]

```
class Watch {
Color color;
Waterproof waterproof;
Depth depth;
...
}
class Color {...}
class Yellow extends
Color {...}
class Metallic extends
Color {...}
class Depth {...}
class 50m extends
Depth {...}
class 100m extends
Depth {...}
```

[S13]

```
// The syntax of
feature tags is:

<fcomment> ::=
"/*@feature:" <flist>
"@*/" [<filetag>]
<flist> ::=
<featurename> ( ":"
<flist> ) *
<filetag> ::=
"/*@!file_feature!@*/"

// where <featurename>
identifies a
// feature of the FD.
```

[S14]

```
class Test {
     static Exp e;
     public static void main (String args[]) {
            Test.printtest();
            Test.evaltest();

     }

     static void evaltest() {
          e = new Num(1);
          System.out.println("eval (1) = " +
e.e
          e = new Neg (new Num (1);
          System.out.println("eval(Neg(1) ="
          e = new Plus (new Num(1), new
Num(2));
          System.out.println("eval(1+2)=" + e.e
          e = new Neg(new Plus(new Num(1), new
Num
          System.out.println("eval(-(1+2))=" +
     }

     static void printtest() {
          e = new Num(3);
          System.out.println("print(3) = " + e)
          e = new Neg(new Num(5));
          System.out.println("print(Neg(5)) = "
          e = new  plus(new Num(5), new
Num(7));
          System.out.println("print (5+7) = " +
     }
```

[S35]

```
Relationships: 2 sequence { 3 signal_quality 4
cloneBetween 0 and 5 vehicle 5 //. . . 6 } 7 8
Attributes: 9 @NT string sequence.comment 10 @RT int
vehicle.speed [0..130] delta 5 default 4 0 11 @ND
int *.cost [0 .. 1000] default 150 12 real
signal_quality.luminance_mean 13 [0.0 .. 3 2.0]
delta 2.0 14 [3 2.0 .. 224.0] delta 8.0 15 [224.0 ..
255.0] delta 2.0 16 default 7 2.5 5 17 //. . . 18 19
Descriptions: //. . . 20 21 Constraints: //. . . 22
23 Objectives: 24 objective
generate_low_cost_configurations { 25 min (sum
(*.cost)) 26 } 27 Configurations: //. . .
```

Table XIII. Results of the Quality Assessment of the Primary Studies

	No (0)	Partial (0,5)	Yes (1)	Average Score
Q1	1	2	34	0.95
Q2	12	10	15	0.54
Q3	1	14	22	0.78
Q4	23	12	2	0.22
Q5	11	15	11	0.50
Q6	19	7	11	0.39
Q7	16	15	6	0.36
Q8	23	8	6	0.27

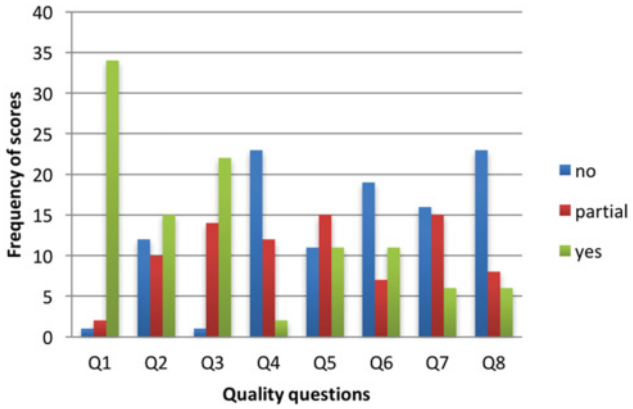


Fig. 10. Frequency analysis of quality scores for each question.

Table XIV. Quality Attributes Addressed by Studies

	Does Not Mention (0)	Mentions (1)	Contribution (2)	Contrib. and Eval. (3)	Average Score
Usability	19	8	10	0	0.76
Integration	21	0	13	3	0.97
Scalability	24	8	4	1	0.51
Performance	24	4	8	1	0.62

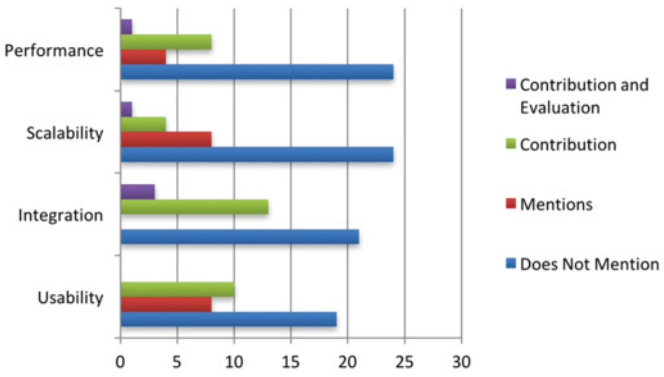


Fig. 11. Frequency analysis of scores for each quality attribute.

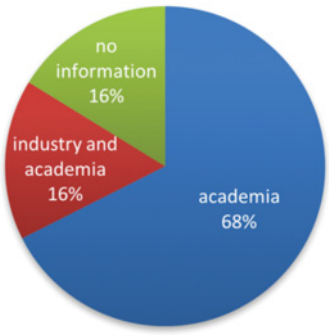


Fig. 12. Research context of primary studies.

Few evaluate integration and only one study evaluates scalability and performance, respectively. Even though quite a few tools contribute to the various quality attributes, the lack of attention of researchers to these quality attributes (not explicitly mentioning them or evaluating them), which are high up in the priority list of practitioners, can be seen as another potential reason behind the very limited industrial adoption of these tools.

4.4. RQ4: What is the Context of Research?

This section discusses the context of the research in the surveyed studies, i.e., whether the research is conducted in an academic or industrial context, the industrial domain of focus, and the covered main activities in a product line approach (scoping, analysis, implementation, testing).

4.4.1. Research Context: Academia vs. Industry. The distribution of the research context of the studies is presented in Figure 12. The figure shows that most studies (68%) have been conducted in an academic context. Only 16% of the studies are joint industrial-academic endeavors. In 16% of the studies, no information was provided on the research context. Table XV presents a list of all studies with their research context.

Although the primary research context of some studies was academic, few still had practical relevance. Figure 13 shows the distribution of the relevance of the primary studies. Almost half of the studies (41%) are relevant to academia only. 36% of the studies are relevant to both academia and industry, with 10% of the studies relevant to practice only. Finally, 13% of the studies provided no sufficient data to be classified.

4.4.2. Focus and Industrial Domain. In terms of a potential specific industrial application domain, most of the surveyed studies (86%) can be considered as generic, i.e., not focusing in a particular sector. Just 14% of the studies are specific enough to be linked to a particular domain (see Table XVI). Furthermore, the application domains of the examples presented in the studies were quite diverse (see the rightmost column of Table XVI).

4.4.3. Covered Product Line Activities. We analyzed the surveyed studies with respect to the supported product line activities. It is often not a clear-cut decision whether an approach supports a particular activity. For instance, many approaches for product configuration assume that configuration decisions are made by considering requirements in some way. Additionally, it is difficult to clearly distinguish the activity of scoping (defining the boundaries of a product line and its targeted market segment) from domain analysis and the process of defining the available configuration options.

Table XV. Research Context of the Primary Studies

	Academia	Industry and academia	No information
S1	X		
S2	X		
S3	X		
S4	X		
S5	X		
S6	X		
S7	X		
S8	X		
S9			X
S10			X
S11	X		
S12		X	
S13		X	
S14	X		
S15	X		
S16	X		
S17	X		
S18	X		
S19		X	
S20	X		
S21			X
S22	X		
S23	X		
S24			X
S25			X
S26	X		
S27	X		
S28	X		
S29	X		
S30			X
S31	X		
S32	X		
S33		X	
S34		X	
S35		X	
S36	X		
S37	X		

Fig. 13. Relevance of primary studies.

Table XVI. Industrial Domain of the Primary Studies

	Focus	Industrial domain	Domain of example
S1	Domain analysis	Generic	—
S2	Domain analysis	Generic	—
S3	Conceptual model of feature-oriented programming	Generic	Calculators, military simulators
S4	Variability realization	Generic	Computer aided dispatch systems
S5	Variability modelling	Generic	Factory automation, e–commerce
S6	Feature modelling	Generic	—
S7	Variability modelling	Generic	Product line of watches
S8	Variability modelling	Generic	Intelligent traffic systems
S9	(Consistency in) Variability modelling	Generic	—
S10	Variability modelling	Generic	—
S11	Variability modelling	Generic	—
S12	Variability modelling	Generic	Steel plants
S13	Variability realization	Generic	Satellites
S14	UI for Variability realization	Generic	—
S15	Variability modelling	Generic	Graph library
S16	Feature modelling	Generic	—
S17	Variability modelling	Generic	Surveillance cameras (toy example)
S18	Configuration	Generic	Automotive, weather station network
S19	Configuration, generation of products	Web-based applications	Media
S20	Scoping	Generic	Planning software (fictitious)
S21	DSPL for self-adaptive systems	Generic	—
S22	Testing of feature model analysis techniques	Feature model analysis techniques	—
S23	Survey of Tools	Generic	—
S24	Configuration	Generic	Automotive software (fictitious)
S25	Software Product Line Engineering	Generic	Service robots
S26	RE for product lines (i.e., combined with feature modeling)	Generic	—
S27	Domain analysis	Generic	Electronic submission system
S28	Configuration	Linux kernel	—
S29	Configuration, Optimization of configuration	Mobile applications	—
S30	Configuration	Generic	—
S31	Configuration, product derivation	Generic	Yard management system
S32	Configuration, Optimization of configuration	Generic	—
S33	(Introducing variability to) Model-based Testing	Generic	Aerospace
S34	Analysis of Variability	Generic	—
S35	Product derivation in specific domain (video)	Generation of customized videos	—
S36	Model checking of product lines	Generic	Bike sharing
S37	Synthesis of feature models	Generic	Wikis
		Generic: 32 out of 37 (86%)	
		Specific domain: 5 out of 37 (14%)	

Table XVII. Covered Product Line Activities

	Scoping and Requirements	Domain Analysis(incl. Definition of a Configuration Space)	Domain Implementation	Testing and V&V(excl. Consistency of Artifacts)
S1	(x)	x	—	—
S2	(x)	x	—	—
S3	—	x	x	—
S4	—	(x)	x	—
S5	x	x	x	—
S6	—	x	—	—
S7	—	x	x	—
S8	—	x	x	—
S9	—	(x)	x	—
S10	x	x	(x)	—
S11	—	x	—	—
S12	—	x	x	—
S13	—	—	x	—
S14	—	—	x	—
S15	x	x	x	—
S16	—	x	—	—
S17	x	x	(x)	—
S18	—	x	—	—
S19	(x)	x	x	—
S20	x	(x)	—	—
S21	—	(x)	—	—
S22	—	x	—	x
S23	—	x	x	—
S24	—	x	—	—
S25	(x)	x	x	x
S26	—	x	—	—
S27	x	x	(x)	—
S28	—	x	—	—
S29	—	x	—	—
S30	—	x	—	—
S31	—	x	x	—
S32	—	x	—	—
S33	—	x	(x)	x
S34	—	x	x	—
S35	—	x	x	—
S36	—	(x)	—	x
S37	—	x	—	—
Total	**10**	**35**	**20**	**4**
Percentage	**27%**	**95%**	**54%**	**11%**

x = fully covered, (x) = partly covered, — = not covered.

Table XVII shows an overview of which approach supports which product line activity. We distinguish between “Scoping and Requirements”, “Domain Analysis”, “Domain Implementation”, and “Testing/ Verification & Validation”. It is worth noting that “Domain Analysis” includes activities leading to the definition of a configuration space and the available options. “Testing/V&V” addresses the quality of the products and their implementation, e.g., testing products for defects. This does not include techniques addressing the consistency between artifacts.

Figure 14 provides the corresponding overview: 27% of the studies can be considered addressing aspects of “Scoping and Requirements”. Not surprising, given the scope of this survey, almost all studies (95%) address Domain Analysis in one way or another.

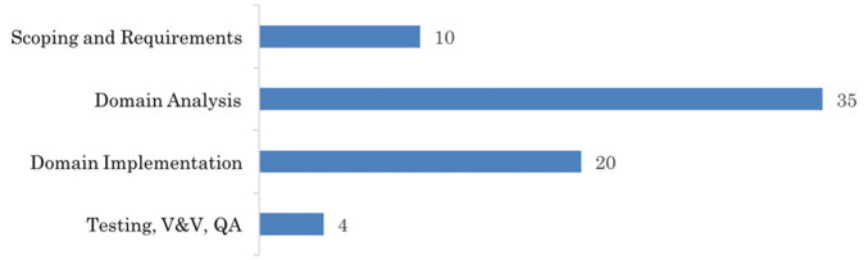


Fig. 14. Covered product line activities.

54% of studies consider aspects of "Domain Implementation", and only 11% address "Testing/Verification & Validation".

4.5. RQ5: What are the Main Challenges Faced by Current Product Line Management (PLM) Tools?

Our last research question aimed at analyzing the main challenges faced by current tools as well as limitations of the tools. We, therefore, analyzed the 37 selected studies regarding the challenges and limitations of current variability management tools they discuss. Using the coding technique [Seaman 1999], we first scanned the studies looking for keywords "challenge", "issue", "limitation", and "drawback" and then extracted the related text (statements on challenges and/or limitations). This allowed us to find out which studies do not discuss any limitations or challenges (no statements extracted); which studies at least mention challenges or limitations (statements extracted list challenges or limitations, but do not discuss them); and which studies actually discuss challenges or limitations (statements extracted list and discuss challenges or limitations). 56% do not discuss limitations at all, 27% at least mention some limitations without further discussing them, and only about 17% actually discuss limitations. We find this a general weakness of publications on variability management tools, i.e., that they do not discuss their own limitations, which makes it hard to assess tools' usefulness.

Challenges are more frequently discussed (73% provide a discussion, 13% at least mention challenges, only 13% do not even mention challenges), i.e., authors mention what was the challenging part of implementing their tool and/or what challenges their tool addresses.

We eventually analyzed the extracted statements and (through discussion and refinement among researchers) came up with ten categories for challenges and limitations, in which we could group the extracted statements on challenges and limitations discussed in detail in the following text (ordered by the number of studies providing input to the category).

The key challenge of variability management tools is *scalability of models*, i.e., how to develop variability models that are still useful despite their size and complexity. 40% of the selected studies discuss this challenge and suggest different solutions as described earlier. The second most discussed challenge is *checking models for consistency and correctness* (23%), especially how to keep the models consistent with the underlying architecture and check that the models represent the variability of the product line correctly. *Mapping problem and solution space* (20%) is also discussed as a key challenge to be addressed by variability management tools. Many tools only take care of creating and managing the variability models representing variability but not of how to map variability (e.g., represented by features or decisions) with the actual artifacts realizing this variability. *Visualization / Graphical* Overload is discussed as a challenge by 17% of

the selected studies. Variability management tools must provide ways to cope with the size and complexity of variability models to help users suffering from graphical overload with visualizations. Other important challenges are *usability* and *maintenance and evolution of variability models* (both 13%). Addressing both challenges is essential for tools to be useful and successful in practice in the long run. *Integration of variability management and (legacy) software (development)*, i.e., the question of how to adopt a variability management tool in practice, is also still an important issue and discussed by 10% of the selected studies. *Process Improvement/Automation through variability management* (7%) is explicitly discussed by 2 selected studies, even though this is actually the key goal of variability management tools anyway. Two further challenges, which are discussed by one study each, are supporting the modeling of *non-functional properties in variability management (e.g., resource consumption constraints) and compliance (with standards/quality policies/regulations)*.

4.5.1. Scalability of (Variability) Models (12 Studies). In an initial discussion, we had called this category "working with one large model vs. working with several separate models". However, through our discussion we found out that the statements we categorized here actually are all about challenges regarding the scalability of (variability) models.

For instance, the authors of [S17] report experiences from empirical case studies that confirm that the complexity of variability management stems from the need to work with (too) large models. Study [S4] highlights the importance of compositional approaches to product line representation/implementation to address this challenge. Study [S21] report on a tool supporting variability management in self-adaptive systems, which again adds to the challenge of scalability of models.

As discussed by the authors of [S3] a key "challenge is to show how scaling can be accomplished in a principled manner so that product line variability management tools are not just ad-hoc collections of tools using an incomprehensible patchwork of techniques". More specifically, they argue that "generators are a technological statement that the development of software in a domain is understood well enough to be automated. However, we must make the same claim for generators: The complexity of generators must also be controlled and must remain low as application complexity scales; otherwise, generator technology will unlikely have wide-spread adoption."

The BVR tool [S7], for instance, proposes to have separate models related to a base model instead of one large model or completely separate models to allow working with product lines of a realistic size. DOPLER [S12] allows both, creating one big model and several small but related models. The DSL tool FAMILIAR [S30] suggests separating, relating, and composing several feature models while automating the reasoning on their compositions. FAMILIAR focuses mainly on textual representation because, as they claim, this favors readability of the specified operations and leads to more usability and productivity when dealing with compositional operations on feature models. They, however, also argue that graphical visualization has proved to assist users, for example, during the configuration process. This is why they integrated their DSL with the Feature IDE tool. The author of [S11] presents a NUI-based multiple perspective variability modeling tool to help working with large-scale models, i.e., multi-touch interfaces to allow working with large models (and their visualizations/different views) to address the scalability challenge. ViewInfinity [S14] provides seamless and semantic zooming of different abstraction layers of an SPL. The tool described in [S5] provides multiple product line views (using the feature model as a unifying view). Study [S8] focuses on the hierarchical organization of variability, the first class representation of simple and complex dependencies (*"dependencies that affect the binding of a large number of variation points, e.g., quality attributes"* [S8]); and argues that relations between dependencies should be explicitly represented. The Odyssey Reuse environment

[S10] specifies *"patterns based on both architectural styles and specific information from the application domain to create a complete reuse environment, which defines software architectures and conceptual model representations on a high level of abstraction."*

4.5.2. Checking Models for Consistency and Correctness (7 Studies). Checking the models underlying the variability management tools for consistency and correctness is considered as a key challenge by seven of the 30 studies. For instance, the authors of RequiLine [S26] argue that semantic information is needed for an automated consistency check in variability management tools. Study [S5] highlights that consistency checking among the multiple views in a product line (as provided by their tool) is essential. FeatureMapper [S9] provides diverse visualizations to support the SPL engineer in verifying the correctness of the models (feature models, mapping models, solution space models) and argues this is very important. The authors of Odyssey [S10] suggest specifying the *"operations that will be performed on models, as well as to systematize these operations, to facilitate the consistent creation of models."* The DOPLER tools [S12] have an integrated consistency checking component that checks the consistency on different levels, i.e., in problem space, in solution space, and between problem and solutions space. ToolDAy [S27] is one of the few studies that discuss their limitations, i.e., that complex consistency rules cannot be described in their tool. The authors of study [S3] highlight the use of model checkers in their tool as important future work.

4.5.3. Mapping Problem and Solution Space (6 Studies). Six studies highlight the challenges and limitations of mapping problem and solution space, i.e., mapping the variability representation with the actual product line architecture. For instance, ISMT4SPL [S17] discusses "traceability between decisions in variability/feature models and the corresponding implementation artifacts" as a key challenge for variability management tools. The authors of study [S16] report about a limitation of their tool, i.e., that the support for mapping problem and solution space is missing. FeatureMapper [S9] explicitly focuses on this aspect by introducing mapping models to map feature models and solution space models. Kumbang [S18] explicitly integrates architecture models (i.e., Koalish, an architecture description language/component model based on Koala ADL but adding variability concepts) with feature models within its tool support. DOPLER [S12] uses explicit asset models to represent the solution space and links these models with the problem space decision models via so-called inclusion conditions. Code tagging tools such as XToF [S13] do not map both spaces but rather integrate the representation of the problem space into the solution space, or, as could be argued, just represent solution space variability (i.e., variability in code).

4.5.4. Visualization/Graphical Overload (5 Studies). Five studies argue that visualization of variability easily leads to a graphical overload of the tool user and is a key challenge. For instance, the author of study [S11] argues that *"it is important for a variability management mechanism to be able to extract and present relevant information about a variability model in dedicated views for different groups of stakeholders (users, system analysts, developers, etc. to alleviate the graphical overload when showing all the information in one view."* ViewInfinity [S14] provides seamless and semantic zooming of different abstraction layers of an SPL. Study [S8] argues that variability models should *"represent variation points as first class entities in all abstraction layers (from features to code); provide a hierarchical organization of variability; focus on the first class representation of simple and complex dependencies (dependencies that affect the binding of a large number of variation points, e.g. quality attributes); and explicitly represent dependencies"*. ST2T [S24] provides sophisticated visualization and interaction techniques to address the challenge that handling variability and configurations is hard due to the complexity on a cognitive level as human engineers reach their limits

in identifying, understanding, and using all relevant details. Study [S16] highlights this as a key limitation of their tool, i.e., that a graphical representation missing.

4.5.5. Maintenance and Evolution of Variability (Models) (4 Studies). Four studies report on the challenges and limitations regarding maintenance and evolution of variability (models). The BVR tool [S7] suggests to not use annotations of features but *"relations between feature models and elements of a base model"* to express/capture variability. Study [S30] confirms that with current technologies manipulating and evolving large-scale feature model is challenging and error-prone. Study [S29] argues that not all devices and their characteristics can be known in advance—*"their unique capabilities must be discovered and dealt with efficiently and correctly"*. Study [S6] reports that ambiguities in existing feature meta-models negatively affect maintenance.

4.5.6. Usability (4 Studies). Only one study, RequiLine [S26], mentions usability to be a limitation of their tool support. However, most tools suffer from this limitation in our own experience. The authors of [S4] admit that the understandability of their variability modeling language/tool must be improved. DOPLER [S12] puts a special emphasis on usability, however, only on the configuration side, i.e., the configuration tools are optimized to allow their use by sales staff. ST2T [S24] provides sophisticated visualization and interaction techniques to make complex variability models usable by engineers.

4.5.7. Integration of Variability Management and Legacy Software (3 Studies). Three studies report about the challenge of integrating variability management support into legacy software. The development of XToF [S13], for instance, was motivated by industrial needs. One of the key goals was to develop support for variability management that does not require changing current development practices in the organization requesting support. Thus, a code-tagging approach was applied. The authors argue that it is important to provide tool support for variability management, but this support must be nicely integrated with existing tools and processes. The development of FeatureIDE [S15] was challenged by the difficulty to integrate variability management and Eclipse. The author of the ToolDAy [S27] argues that supporting integration with tools like DOORS is essential (though not supported by ToolDAy).

4.5.8. Process Improvement/Automation through Variability Management (2 Studies). Two studies describe the challenge of improving development processes through automation provided by variability management tools. The authors of study [S19], for instance, argue that "on the one hand, the non-existence of a unified way to introduce the contents [leads to] an unnecessary waste of time for the employees to learn new technologies and feel comfortable with the new platforms. On the other hand, a rapid prototyping platform is also desirable for showing their customers a working prototype at an early stage." The authors of study [S1] highlight the need for models that are expressive enough for automation.

4.5.9. Compliance (with Standards/Quality Policies/Regulations) (1 Study). Study [S13] stresses the need for compliance, i.e., they argue that it is also important that variability management/modeling tools do not violate with standards/quality policies/regulations in the organizations in which they are used.

4.5.10. Non-Functional Properties in Variability Management (1 Study). Study [S29] argues that resource consumption constraints are not taken into account by existing configuration approaches and tools.

5. COMMERCIAL TOOLS AND TOOL ADOPTION IN INDUSTRY

In addition to our SLR, we conducted a web search on commercially available variability management tools as well as studies on tool adoption in industry/practice. In this section, we briefly discuss our findings under RQ1 and RQ2 (cf. Sections 2 and 4). While RQ3 and RQ4 are irrelevant to this section, RQ5 cannot be addressed due to the lack of primary studies reporting challenges in relation to these tools (and our inability to provide our own assessment in a secondary study such as an SLR).

5.1. Commercial Variability Management Tools

5.1.1. RQ1: What tools have been developed to manage variability in software product lines?. We explicitly focus on tools developed to support variability management in software product line engineering, thus leaving out commercial tools developed in other communities such as the CWAdvisor [Felfernig et al. 2001] or the SAP Configurator [SAP Configurator 2016], which follow an AI-based process or MetaEdit+ [Arion and Tolvanen 2004], which is a domain-specific language and code generation environment. Some industries have extended other commercial tools with support for variability management [Berger et al. 2013], typically without following a particular product line engineering process. For example, IBM Rational DOORS [IBM Rational DOORS] comes with a requirements management add-on that supports the definition of variability within requirements documents. Another example is SparxSystems Enterprise Architect [SparxSystems Enterprise Architect 2016] which has also been extended with variability management support. Another common industrial practice is the use of Microsoft Excel or Microsoft Word to document variability.

All these solutions or practices to variability management work very well within the context they were developed for; however, they do not follow any particular product line engineering approach. We were only able to identify two commercial tools developed for product line variability management, namely, pure::variants [Beuche 2016] and Gears [Krueger and Clements 2014].

5.1.2. RQ2: What are the Characteristics of These Tools? pure::variants [Beuche 2016; Pure-Systems 2016] is developed by pure-systems GmbH in Magdeburg, Germany. The tool supports variant management and product configuration based on feature models and has a strong focus on interoperability and extensibility. For example, the tool can be integrated in the Eclipse IDE, used with a web browser, as a command line client, and even in a custom application. Several extensions to existing commercial-off-the-shelf tools exist, e.g., to DOORS or SAP. Four types of models can be created and managed with pure::variants: (1) Feature Models that represent the variability within a system; (2) Family Models that represent the variants of assets that can be selected; (3) Variant Description Models that are used to store the selected features and their values; and (4) Result Models based on 1–3 that represent one concrete instance derived from a product line. Constraints on model elements can be defined using a comprehensive dialect of the language prolog. A Prolog-based constraint solver then allows validating selected configurations. The main benefits of pure::variants are: (i) the strong focus on interoperability and extensibility; (ii) the high number of available extensions; and (iii) the comprehensive support for model checking and validation (also during product configuration). There have been various reports of successful deployment of pure::variants in industry.

Gears [Krueger and Clements 2014; BigLever 2016] is a commercial tool developed by BigLever Software Inc., Austin, Texas, USA. The tool has been developed in Java and supports the three-tiered methodology proposed by Krueger [2007]. The tool allows defining arbitrary reusable software assets and a product feature profile that describes products in terms of features. Gears focuses on products, where feature profiles define

the products that can be built from assets and the optional and alternative choices that can be made for each product. Product configuration is supported by the Gears Configurator, which automatically assembles and configures assets to produce products based on feature choices made using feature profiles. Gears can be tailored to different environments with parameter sets representing different kinds of variability. Dependencies are modeled as global constraints that are checked during configuration. The main benefits of Gears are: (i) its strong product focus; (ii) the possibility to use arbitrary assets; and (iii) its structured foundation based by the three-tiered methodology. As with pure::variants, there are various reports of successful adoption of Gears in industry.

5.2. Wider Tool Adoption in Industry

Djebbi et al. [2007] report findings of a study on the ability of product line management tools to answer industry needs. They identified 12 tools through an unsystematic search (we cover most of them on our SLR) but only analyzed four tools in detail based on their availability. These four tools were RequiLine [S26], pure::variants [Beuche 2016], XFeature [S6] and DOORS-TREK (an add-on to IBM Rational DOORS) [DOORS-TREK 2013]. Djebbi et al. [2007] describe these tools and discuss the support of these tools for variability modeling as well as the support for management (such as reporting capabilities) they provide. They conclude that tools developed in industry or in industry projects work well for the context they have been developed for but are hard to apply in other contexts.

Berger et al. [2013] report the results of a survey on variability modeling in industrial practice. Among other questions, they asked industrial practitioners what variability modeling tools they use. Respondents could select from 10 particular tools or specify an open answer. pure::variants [Beuche 2016] was the most used tool, followed by Gears [Krueger and Clements 2014]. From the tools we identified in our SLR, FeatureIDE [S15], DOPLER [S12], X-Feature [S6], and AHEAD [S3] were the only ones mentioned by respondents. This confirms our findings on the difficulty of research tool adoption in industry. As Berger et al. [2013] conclude "all other tools play only a minor role in the participating projects" and were only reported as being used once or twice. The answers of the 42 survey respondents were analyzed in detail and it was found that many respondents use "other open source tools", "other commercial tools", or "home-grown domain-specific tools." A key finding regarding variability modeling tool support of the survey was that there exists a wide variety of home-grown solutions developed in industry that are unknown to researchers.

Lettner et al. [2013] confirm the findings of Berger et al.'s survey by reporting that industry often develop custom solutions to automate the configuration process of their variable software systems. These solutions are often not based on variability models but describe configuration knowledge directly in code or in simple XML files. Comparing a custom-developed with a model-based configuration approach led them to the conclusion that using a model-based solution could be beneficial to industry. For instance, it would help to decouple configuration UI and variability information and make the approach more adaptable and extensible.

6. STUDY LIMITATIONS AND THREATS TO VALIDITY

In this section, we discuss the limitations of this study and any threats to its validity. Of particular importance are some of the inherent limitations of the adopted SLR research methodology [Kitchenham et al. 2009] and its execution.

The first limitation is the comprehensiveness of the search process and the threat that some relevant primary studies, and subsequently tools, might have been missed. This could be due to the varying use of terminology among different communities, which might not have been captured by the formulated search string (e.g., approaches that

use constraint programming to represent configuration problems). To address this, we extended the traditional SLR search protocol and introduced a number of mitigating measures. First, and in addition to searching main publisher websites as discussed in Section 2, we ran our search string on multiple general indexing search engines, such as Google Scholar and Scopus, to ensure the widest coverage possible, especially given that different search engines use different search and ranking algorithms. As described earlier, we found a total of 556 papers, which were first analyzed by one researcher to exclude non-SPL related papers. To address the threat that important publications might have been excluded in this step, forward and backward reference checking was used on the remaining publications to try and follow any potential leads to related work that might have been missed. Finally, manual searches were conducted on the proceedings of known outlets in the area and on the outputs of active researchers in the field.

Second, and as with all SLRs, our inclusion and exclusion criteria stipulated that primary studies had to be peer reviewed to qualify. This meant that gray literature, theses, and commercial tools, which are not covered by peer-reviewed publications (providing enough details about the tool), were not included. Even though some SPL tools, such as the ones from BigLever and Pure::Systems, have also been published in peer-reviewed papers [Krueger and Clements 2014; Beuche 2016], they tend to be short tool demonstration papers. This is understandable given the interest in protecting the intellectual property embedded in the technical details of such tools. However, to assess the impact of this on our conclusions, we conducted a comprehensive search on commercially available tools (see Section 5) and could not identify more than two tools (following a product line engineering process) and a few publications discussing the practical use of tools in industry [Berger et al. 2013; Krueger 2007; DOORS-TREK 2013]. Given the study examined thirty-seven tools, the impact on the analysis and conclusions, had the commercial tools been included, would have been limited.

Finally, we only covered literature published in English. This would automatically put us at risk of missing tools published in other languages. Although we had limited options to address this issue due to the language barrier, we believe that it is currently highly unlikely for significant research to remain unpublished in English for long.

Beyond the above limitations impacting the scope and completeness of the dataset upon which our analysis was conducted, threats to the analysis process and the conclusions drawn can be discussed under four main headers, *mainly construct, internal, external* and *conclusion* validity [Matt and Cook 1994].

Construct and *internal* validity relate to the robustness of the implementation of the research methodology adopted, which is the SLR methodology in our case. Some of these threats have already been discussed in this section such as the completeness of the search process. Another important validity problem in SLRs is author bias. This has been acknowledged and addressed from the onset of this work given that a number of the researchers involved in this study are active in the research area and produced related tools. Accordingly, specific measures were put in place to eliminate any potential bias. This included having multiple reviewers for each primary study. When considering a primary study that was authored by any of the researchers involved this study, these researchers were excluded from the decision-making (for inclusion/exclusion) and data extraction related to their own work. An external researcher (not one of the authors) then validated the overall review process to ensure its independence before it was applied.

Finally, *external* validity relates to the applicability of the results of the study beyond its initial scope and *conclusion* validity relates to the robustness of the conclusions drawn. One of the main threats here is potentially influencing the analysis process in

order to produce conclusions that are aligned with the researchers' initial hypothesis and views. In our case, this was not an issue as the work started with open research questions without any pre-formulated stance. Conclusions made were all based on grounded theory [Martin and Turner 1986] and were reported along with the data upon which the analysis was based.

7. RELATED WORK

There is a number of systematic studies (i.e., systematic literature reviews and systematic mapping studies) focusing on various aspects of software product lines. Table XVIII provides an overview of such studies. These studies often do not consider tools at all or consider tool-support as one of multiple aspects (see the rightmost column in Table XVIII).

The closest to our work is the survey by Lisboa et al. [2010] on domain analysis tools. They survey 19 tools and cover up to 2005. Our coverage is more extensive, both in terms of surveyed tools as well as the covered period.

8. CONCLUSIONS AND FUTURE RESEARCH DIRECTIONS

8.1. Summary

We reported on a survey in which we analyzed 37 variability management tools identified using a systematic literature review to understand the tools' characteristics, maturity, and the challenges in the field. The tools are based on diverse development environments, apply diverse technologies, and support different variability modeling approaches. Most tools support a feature modeling approach. Different graphical and textual notations are provided by the tools, with a focus on tree-based visualizations of features. Only few tools provide multiple views, e.g., a graphical view of features together with a text-based representation of source code variability. While most studies about variability management tools provide a good motivation and a description of the research context they often lack data, e.g., from empirical studies with tool users, to support the claims made and the findings reported. Also, studies seldom provide a critical reflection of the presented tools and their limitations. Most variability management tools were not well evaluated, especially with respect to feedback from end users. Quality attributes important for the practical use of tools such as usability, integration, scalability, and performance are out of scope for most of the analyzed studies. This might be explained by the fact that most studies have been conducted in an academic context. Only 3 out of the 37 studies were joint industrial-academic endeavors.

8.2. Future Research Directions

The studies analyzed in this survey discuss particular challenges related to their focused aspect of variability management. We have performed a comprehensive analysis of these challenges and reported on the details in Section 4.5. From these challenges, we identified few potential research directions that require further investigation.

8.2.1. Scalability and Complexity. A challenge that has been commonly reported is the *scalability and complexity of models*. This can occur in various forms, e.g., when handling one very large model or groups of multiple smaller models, with dependencies between them. The resulting research challenges can be roughly clustered into two groups: (1) *computational complexity/scalability* (tools and techniques to handle very large models, performance of automated mechanisms) and (2) *cognitive complexity/scalability* (handling of very large models by humans, interacting with large models, comprehending large models and the consequences of changes/decisions). For the first group, computational challenges, research on techniques for processing very large

Table XVIII. Overview of Systematic Studies in Product Line Engineering and Their Coverage of Tool-oriented Aspects

Reference	Year of publ.	Type of study	Focus (in a product line context)	Covered period	Tool-oriented aspects
[Alves et al. 2010]	2010	SLR	Requirements engineering	1990–2009	Availability of RE tools
[Bakar et al. 2015]	2015	SLR	Feature extraction from natural language	2005–2014	(Brief section on) tool support in approaches for feature extraction
[Castelluccia and Boffoli 2014]	2014	Map	Service-oriented SPL	2008–2012	—
[Chen and Babar 2011]	2011	SLR	Variability management	1990–2007	—
[da Mota Silveira Neto et al. 2011]	2011	Map	Testing	1993–2009	(Brief section on) test automation tools
[da Silva et al. 2011]	2011	Map	Agile SPL	2005–2010	"Tool support" as 1 out of 17 practice area facets based on the SEI framework
[dos Santos Rocha and Fantinato 2013]	2013	SLR	Business process management	2003–2012	—
[Engström and Runeson 2011]	2011	Map	Testing	2001–2008	"Tool" as 1 out of 5 contribution types
[Guedes et al. 2015]	2015	Map	Variability management in DSPL	2006–2015	"Tool support" as criterion (yes/no)
[Holl et al. 2012]	2012	SLR, ES	Multi product lines	1999–2010	"sharing and deploying product line models and tools" as 1 out of 7 reported capabilities(Brief section on) "Tool support for modeling multi product lines"
[Khurum and Gorschek 2009]	2009	SLR	Domain analysis	1998–2007	—
[Laguna and Crespo 2013]	2013	Map	PL evolution, re–engineering, and refactoring	1998–2011	Tools for model extractionTools for re-engineering of legacy codeTools in SPL-related refactoring papersMapping of considered aspects (in reengineering legacy systems and SPL refactoring) to tools
[Lisboa et al. 2010]	2010	SLR	Domain analysis tools	1993–2005	Survey of 19 domain analysis tools
[Lopez-Herrejon et al. 2015a]	2015	Map	Testing (Combinatorial Interaction Testing, CIT)	2006–2014	Names identification of "techniques, algorithms, and tools used for CIT in SPLs" as goal, but does not address tools specifically.

(Continued)

Table XVIII. Continued

Reference	Year of publ.	Type of study	Focus (in a product line context)	Covered period	Tool-oriented aspects
[Lopez-Herrejon et al. 2015b]	2015	Map	Search-based SE	2001–2014	"Tool support" as 1 out of 10 classification criteriaDiscussion identifies "need for better tooling support"
[do Carmo Machado et al. 2014]	2014	SLR	Testing	1998–2013	Discusses whether tool support as available (for selection of products to test, to handle test of end-product functionalities)
[Mohabbati et al. 2013]	2015	Map	Service-oriented SPL	2000–2011	"Tooling support" as 1 out of 8 contribution type
[Montagud et al. 2012]	2012	SLR	Quality attributes	1996–2010	"Tool support" as criterion (Manual/Automatic)
[Myllärniemi et al. 2012]	2012	SLR	Quality attributes	2000–2010	—
[de Sousa Santos et al. 2015]	2015	Map, Ex	Textual use case	2003–2014	Identify development of tool support as future work
[Sepúlveda et al. 2016]	2016	SLR	Requirements engineering (Modeling of requirements)	2000–2013	"Tool support" as one criterion
[Soares et al. 2014]	2014	SLR	Non-functional properties	2003–2013	Discuss "evidence [as] a means to evaluate the maturity of methods and tools", but do not further discuss tools.
[Vale et al. 2014]	2014	SLR	Bad smells	2007–2013	—

SLR = Systematic Literature Review, Map = Mapping Study, Ex = Experiment, ES = Expert Survey.

models seems to be relevant [Kolovos et al. 2013]. Also, platforms and frameworks that were originally intended for handling large data sets might be interesting.

For the second group, cognitive challenges, work on interactive tools and usability is relevant. As reported earlier, multiple studies mention *visualization and graphical overload* and *usability* as challenges. Here, abstraction is an important means to an end, i.e., the focusing on the information that is relevant for a given purpose or stakeholder concern (and intentionally abstracting away other details). This can be supported by user interfaces and visual representations that are task-adequate. We can address this challenge in multiple ways, for instance by gaining a better understanding of the required SPL tasks in order to build interfaces that are "optimized" for these tasks. Another way is by providing flexible user interfaces, which the users can adapt according to their preferences and tasks. These may even be interfaces that "learn" what is preferred for a particular task.

8.2.2. Consistency Checking and Model Correctness. Another set of challenges where further research is needed is *consistency checking and model correctness*. Here, work from software architecture and consistency between an expected architecture and the structure of the actual implementation seems relevant [Murphy et al. 2001]. Further work in

the context of product lines and variability realization is required. In addition, many of the existing variability management approaches are based on rather simple, i.e., Boolean semantics. When more complex aspects are addressed by the SPL approach, e.g., non-Boolean product properties, ensuring consistency and correctness becomes more challenging and more powerful reasoning approaches need to be applied.

8.2.3. Traceability from Problem to Solution Space. Another group of challenges arise from the need to establish *a mapping between problem and solution spaces*. This is related to the general problem of finding source code that is relevant for a particular concept (e.g., when fixing a particular bug) and techniques for feature location [Dit et al. 2013]. Such techniques are of particular relevance when reverse engineering product lines and establishing a variability representation in the first place (e.g., a feature model and its mapping into implementation artifacts), based on existing products that were not originally implemented using an SPL approach (also see challenges on *integrating variability management and legacy software*).

8.2.4. Maintenance and Evolution. Multiple studies mention challenges around *maintenance and evolution*. Examples are change impact analysis, handling of inconsistencies, change propagation, project management and communication/organizational issues in general within the maintenance team. Addressing such challenges in a product line context is much more complex, e.g., due to more complex artifacts and a higher degree of dependencies among artifacts [Botterweck and Pleuss 2014].

8.2. CONCLUSIONS

Future research directions were based on reported challenges within the various studies analyzed. Thus, some of these challenges could be influenced by the background of the authors. For example, authors with consistency checking background would tend to argue that consistency between product line artifacts is an open challenge. What is, hence, required are more studies that provide empirical evidence on industrial practice in variability management [Berger et al. 2013] and end-user challenges. Here, an industrial context could open up various new commercial perspectives, such as the pressure to get the next product to the market; problems to find the right expert in a large organization; and the challenge of communicating and collaborating within multidisciplinary teams [Rubin and Rinard 2016].

On a "meta" level, when considering the research approaches adopted in the surveyed studies, we observed a wide variation in quality, for example, in terms of: having an explicit research methodology; having clear research questions and evaluation criteria; supporting the replicability of the work; providing a comparison against existing work; and discussing limitations and threats to validity. It is thus recommended that future research pays closer attention to relevant structured research methods, such as the guidelines for conducting and reporting on case studies [Runeson and Höst 2009] and the guidelines for conducting and reporting on experiments [Jedlitschka and Pfahl 2005; Wohlin et al. 2012].

REFERENCES

M. Acher, P. Collet, P. Lahire, and R. B. France. 2011. Managing feature models with FAMILIAR: A demonstration of the language and its tool support. In *Proceedings of the 5th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'11)*. ACM, 91–96.

M. Acher, M. Alférez, J. A. Galindo, P. Romenteau, and B. Baudry. 2014. ViViD: A variability-based tool for synthesizing video sequences. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*. ACM, 143–147.

V. Alves, N. Niu, C. F. Alves, and G. Valença. 2010. Requirements for engineering for software product lines: A systematic literature review. *Inf. Softw. Technol.* 52, 8 (2010). 806–820.

M. Antkiewicz and K. Czarnecki. 2004. FeaturePlugin: Feature modeling plug-in for eclipse. In *Proceedings of the OOPSLA Workshop on Eclipse Technology eXchange (ETX'04)*. ACM, 67–72.

S. Arion and J.-P. Tolvanen. 2004. MetaEdit+: Domain-specific modeling and code generation environment. In *Proceedings of the Workshop on Software Variability Management for Product Derivation - Towards Tool Support*. Springer, Berlin, 1–3.

N. H. Bakar, Z. M. Kasirun, and N. Salleh. 2015. Feature extraction approaches from natural language requirements for reuse in software product lines: A systematic literature review. *J. Syst. Software* 106 (2015), 132–149.

R. Bashroush. 2010. A NUI based multiple perspective variability modeling CASE Tool. In *Proceedings of the 4th European Conference on Software Architecture (ECSA'10)*. Springer, Berlin, 523–526.

D. Batory, J. N. Sarvela, and A. Rauschmayer. 2004. Scaling step-wise refinement. *IEEE Trans. Software Eng*. 30 (2004), 355–371.

G. Bécan, S. B. Nasr, M. Acher, and B. Baudry. 2014. WebFML: synthesizing feature models everywhere. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*. ACM. 112–116.

M. H. ter Beek, F. Mazzanti, and A. Sulova. 2012. VMC: A tool for product variability analysis. In *Proceedings of the International Symposium on Formal Methods (FM'12)*. Springer, Berlin, 450–454.

T. Berger, S. She, R. Lotufo, A. Wąsowski, and K. Czarnecki. 2010. Variability modeling in the real: a perspective from the operating systems domain. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 73–82.

T. Berger, R. Rublack, D. Nair, J. M. Atlee, M. Becker, K. Czarnecki, and A. Wąsowski. 2013. A survey of variability modeling in industrial practice. In *Proceedings of the 7th International Workshop on Variability Modelling for Software-Intensive Systems (VaMoS'13)*. ACM, New York, 7–14.

G. Botterweck and A. Pleuss. 2014. Evolution of software product lines. In *Evolving Software Systems*, T. Mens, A. Serebrenik, and A. Cleve, (Eds.). Springer, Berlin, 265–295.

R. M. M. Braga, C. M. L. Werner, and M. Mattoso. 1999. Odyssey: a reuse environment based on domain models. In *Proceedings of the IEEE Symposium on Application-Specific Systems and Software Engineering and Technology (ASSET'99)*. IEEE. 50–57.

P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil. 2007. Lessons from applying the systematic literature review process within the software engineering domain. *J. Syst. Softw*. 80, 4 (2007), 571–583.

D. 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. 333–336.

BigLever. 2016. Gears. Retrieved from http://www.biglever.com/solution/product.html.

G. Campbell. MTP TOOL- The metaprogramming text processor. Retrieved from http://www.domain-specific.com/index.html.

R. Capilla, A. Sánchez, and J. C. Dueñas. 2012. An analysis of variability modeling and management tools for product line development. In *Proceedings of the Software and Services Variability Management Workshop Concepts, Model and Tools*. 32–47.

D. Castelluccia and N. Boffoli. 2014. Service-oriented product lines: a systematic mapping study. *ACM SIGSOFT Software Eng. Notes* 39, 2 (2014), 1–6.

V. Cechticky, A. Pasetti, O. Rohlik, and W. Schaufelberger. 2004. XML-Based Feature Modelling. In *Proceedings of the International Conference on Software Reuse (ICSR'04)*. Springer, Berlin. 101–114.

L. Chen and M. A. Babar. 2011. A systematic review of evaluation of variability management approaches in software product lines. *Inf. Softw. Technol*. 53, 4 (2011), 344–362.

P. Clements and L. Northrop. 2007. *Software Product Lines: Practices and Patterns* (6th. ed.). Addison-Wesley Longman, Boston, MA.

K. Czarnecki, P. Grünbacher, R. Rabiser, K. Schmid, and A. Wąsowski. 2012. Cool features and tough decisions: A comparison of variability modeling approaches. In *Proceedings of the 6th International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS'12)*. ACM, New York, 173–182.

D. Dhungana, P. Grünbacher, and R. Rabiser. 2011. The DOPLER meta-tool for decision-oriented variability modeling: a multiple case study. *Autom. Software Eng*. 18, 1 (2011). 77–114.

B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk. 2013. Feature location in source code: a taxonomy and survey. *J. Software Evolutio. Process* 25, 1 (2013). 53–95.

O. Djebbi, C. Salinesi, and G. Fanmuy. 2007. Industry survey of product lines management tools: Requirements, qualities and open issues. In *Proceedings of the 15th IEEE International Requirements Engineering Conference*. IEEE. 301–306.

DOORS-TREK. 2013. Retrieved from http://www-01.ibm.com/support/docview.wss?uid=swg24032035.

Eclipse. 2016. Graphical Modeling Framework (GMF, Eclipse Modeling subproject). Retrieved from https://www.eclipse.org/modeling/gmp/.

H. Eichelberger, S. El-Sharkawy, C. Kröher, and K. Schmid. 2014. EASy-producer: product line development for variant-rich ecosystems. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*. ACM. 133–137.

E. Engström and P. Runeson. 2011. Software product line testing - A systematic mapping study. *Inf. Softw. Technol*. 53, 1 (2011), 2–13.

M. Eysholdt and H. Behrens. 2010. Xtext: implement your language faster than the quick and dirty way. In *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion (OOPSLA'10)*. ACM. 307–309.

A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker. 2001. Intelligent support for interactive configuration of mass-customized products. In *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE'01)*. Springer, Berlin. 746–756.

R. Fernández, M. A. Laguna, J. Requejo, and N. Serrano. 2009. Development of a feature modeling tool using microsoft DSL tools. Technical Report. University of Valladolid. GIRO Technical Report 05/01/ 2009.

W. Frakes, R. Prieto-Diaz, and C. Fox. 1997. DARE-COTS: A domain analysis support tool. In *Proceedings of the 17th International Conference of the Chilean Computer Science Society (SCCC'97)*. IEEE. 73–77.

C. Gauthier, A. Classen, M.-A. Storey, and M. Mendonca. 2010. XToF: A tool for tag-based product line implementation. In *Proceedings of the 4th International Workshop on Variability Modeling of Software Intensive Systems (VaMoS 2010)*, ICB-Research Report 37, Universität Duisburg-Essen. 163–166.

H. Gomaa and M. E. Shin. 2004. Tool support for software variability management and product derivation in software product lines. In *Proceedings of the Workshop on Software Variability Management for Product Derivation—Towards Tool Support*. Springer, Berlin. 331–331.

M. Gómez, I. Mansanet, J. Fons, and V. Pelechano. 2012. Moskitt4SPL: Tool support for developing self-adaptive systems. In *Proceedings of the 17th Conference on Software Engineering and Databases (JISBD'12)*.

G. Guedes, C. Silva, M. Soares, and J. B. de Castro. 2015. Variability management in dynamic software product lines: A systematic mapping. In *Proceedings of the 2015 IX Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS'15)*. IEEE. 90–99.

J. van Gurp, J. Bosch, and M. Svahnberg. 2001. On the notion of variability in software product lines. In *Proceedings of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*. IEEE Computer Society, Washington, DC, 45–54.

J. Heer, S. K. Card, and J. A. Landay. 2005. Prefuse: A toolkit for interactive information visualization. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'05)*. ACM. 421–430.

F. Heidenreich. 2009. Towards systematic ensuring well-formedness of software product lines. In *Proceedings of the 1st International Workshop on Feature-Oriented Software Development (FOSD'09)*. ACM. 69–74.

F. Heidenreich, J. Johannes, S. Karol, M. Seifert, and C. Wende. 2009. Derivation and refinement of textual syntax for models. In *Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA 2009)*. Springer, Berlin. 114–129.

A. Hervieu, B. Baudry, and A. Gotlieb. 2011. PACOGEN: Automatic generation of pairwise test configurations from feature models. In *Proceedings of the 22nd Annual International Symposium on Software Reliability Engineering (ISSRE'11)*. IEEE. 120–129.

G. Holl, P. Grünbacher, and R. Rabiser. 2012. A systematic review and an expert survey on capabilities supporting multi product lines. *Inf. Softw. Technol*. 54, 8 (2012), 828–852.

Hydra Feature Modeling. 2009. Retrieved from http://caosd.lcc.uma.es/spl/hydra/index.htm.

IBM Rational DOORS. Retrieved from www.ibm.com.

A. Jedlitschka and D. Pfahl. 2005. Reporting guidelines for controlled experiments in software engineering. In *Proceedings of the International Symposium on Empirical Software Engineering (ISESE'05)*. IEEE. 95–104.

M. F. Johansen, Ø. Haugen, and F. Fleurey. 2012. An algorithm for generating t-wise covering arrays from large feature models. In *Proceedings of the 16th International Software Product Line Conference (SPLC'12)*. ACM. 46–55.

K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson. 1990. Feature oriented domain analysis (FODA) feasibility study. Technical Report. Software Engineering Institute, Carnegie Mellon University CMU/SEI-90-TR-211990.

M. Khurum and T. Gorschek. 2009. A systematic review of domain analysis solutions for product lines. *J. Syst. Software* 82, 12 (2009), 1982–2003.

K. Kim, H. Kim, M. Ahn, M. Seo, Y. Chang, and K. C. Kang. 2006. ASADAL: A tool system for co-development of software and test environment based on product line engineering. In *Proceedings of the 28th International Conference on Software Engineering (ICSE'06)*. ACM. 783–786.

B. Kitchenham, S. Charters, D. Budgen, P. Brereton, M. Turner, S. Linkman, M. Jorgensen, E. Mendes, and G. Visaggio. 2007. Guidelines for performing systematic literature reviews in software engineering. Technical Report. Keele University, UK EBSE-2007–12007.

B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. 2009. Systematic literature reviews in software engineering—A systematic literature review. *Inf. Softw. Technol*. 51, 1 (2009), 7–15.

D. S. Kolovos, L. M. Rose, N. D. Matragkas, R. F. Paige, E. Guerra, J. u. Cuadrado, J. De Lara, I. Rath, D. Varro, M. Tisi, and J. Cabot. 2013. A research roadmap towards achieving scalability in model driven engineering. In *Proceedings of the Workshop on Scalability in Model Driven Engineering*. ACM. 2.

C. W. Krueger. 2007. The 3-tiered methodology: Pragmatic insights from new generation software product lines. In *Proceeding of the 11th International Software Product Line Conference (SPLC'07)*. IEEE. 97–106.

C. Krueger and P. Clements. 2014. Systems and software product line engineering with gears from BigLever software. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*. ACM. 121–125.

M. A. Laguna and Y. Crespo. 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. *Sci. Comput. Program*. 78, 8 (2013), 1010–1034.

D. Lettner, M. Petruzelka, R. Rabiser, F. Angerer, H. Prähofer, and P. Grünbacher. 2013. Custom-developed vs. model-based configuration tools: Experiences from an industrial automation ecosystem. In *Proceedings of MAPLE/SCALE 2013, Workshop at the 17th International Software Product Line Conference (SPLC'13)*. ACM. 52–58.

L. B. Lisboa, V. C. Garcia, D. Lucrédio, E. S. de Almeida, S. R. de Lemos Meira, and R. P. de Mattos Fortes. 2010. A systematic review of domain analysis tools. *Inf. Softw. Technol*. 52, 1 (2010), 1–13.

L. B. Lisboa, V. C. Garcia, E. S. de Almeida, and S. R. de Lemos Meira. 2011. ToolDAy: a tool for domain analysis. *Int. J. Software Tools Technol. Transfer* 13, 4 (2011), 337–353.

R. E. Lopez-Herrejon, S. Fischer, R. Ramler, and A. Egyed. 2015a. A first systematic mapping study on combinatorial interaction testing for software product lines. In *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation (ICST'15) Workshops*. IEEE. 1–10.

R. E. Lopez-Herrejon, L. Linsbauer, and A. Egyed. 2015b. A systematic mapping study of search-based software engineering for software product lines. *Inf. Softw. Technol*. 61 (2015), 33–51.

P. Y. Martin and B. A. Turner. 1986. Grounded theory and organizational research. *J. Appl. Behav. Sci*. 22, 2 (1986), 141–157.

I. do Carmo Machado, J. D. McGregor, Y. C. Cavalcanti, and E. S. de Almeida. 2014. On strategies for testing software product lines: A systematic literature review. *Inf. Softw. Technol*. 56, 10 (2014), 1183–1199.

J. Martinez, C. Lopez, E. Ulacia, and M. del Hierro. 2009. Towards a model-driven product line for web systems. In *Proceedings of the 5th Model-Driven Web Engineering Workshop (MDWE'09)*.

T. von der Maßen and H. Lichter. 2004. RequiLine: A requirements engineering tool for software product lines. In *Proceedings of the 5th International Workshop on Product Family Engineering (PFE'04)*. Springer, Berlin. 168–180.

G. E. Matt and T. D. Cook. 1994. Threats to the validity of research syntheses. In *Handbook of Research Synthesis*. H. Cooper, & L. V. Hedges (Eds.), Russell Sage Foundation, New York.

M. Mendonca, M. Branco, and D. 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. 761–762.

B. Mohabbati, M. Asadi, D. Gasevic, M. Hatala, and H. A. Müller. 2013. Combining service-orientation and software product line engineering: A systematic mapping study. *Inf. Softw. Technol*. 55, 11 (2013), 1845–1859.

S. Montagud, S. Abrahão, and E. Insfran. 2012. A systematic review of quality attributes and measures for software product lines. *Software Qual. J*. 20, 3 (2012). 425–486.

G. C. Murphy, D. Notkin, and K. J. Sullivan. 2001. Software reflexion models: Bridging the gap between design and implementation. *IEEE Trans. Software Eng*. 27, 4 (2001). 364–380.

V. Myllärniemi, T. Asikainen, T. Männistö, and T. Soininen. 2005. Kumbang configurator—A configuration tool for software product families. In *Proceedings of the IJCAI-05 Workshop on Configuration*.

V. Myllärniemi, M. Raatikainen, and T. Männistö. 2012. A systematically conducted literature review: Quality attribute variability in software product lines. In *Proceedings of the 16th International Software Product Line Conference (SPLC'12)*. ACM. 41–45.

P. A. da Mota Silveira Neto, I. do Carmo Machado, J. D. McGregor, E. S. de Almeida, and S. R. de Lemos Meira. 2011. A systematic mapping study of software product lines testing. *Inf. Softw. Technol*. 53, 5 (2011). 407–423

R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. 2000. The Koala component model for consumer electronics software. *IEEE Comput*. 33, 3 (2000), 78–85.

K. Park, D. Ryu, and J. Baik. 2012. An integrated software management tool for adopting software product lines. In *Proceedings of the 11th IEEE/ACIS International Conference on Computer and Information Science (ICIS'12)*. IEEE. 553–558.

A. Pleuss and G. Botterweck. 2012. Visualization of variability and configuration options. *Int. J. Software Tools Technol. Transfer* 14, 5 (2012), 497–510.

K. Pohl, G. Böckle, and F. v. d. Linden. 2005a. Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag, Berlin.

K. Pohl, G. Böckle, and F. van der Linden. 2005b. VARMOD Tool. Retrieved from http://www.sse.uni-essen.de/swpl/SEGOS-VM-Tool/index.html.

Pure-Systems. 2016. pure::variants variant management tool. Retrieved from http://www.pure-systems.com/pure_variants.49.0.html.

T. Quatrani. 2002. *Visual Modeling with Rational Rose 2002 and UML*. Addison-Wesley Longman, Boston, MA.

R. dos Santos Rocha and M. Fantinato. 2013. The use of software product lines for business process management: A systematic literature review. *Inf. Softw. Technol*. 55, 8, 2013. 1355–1373.

J. Rubin and M. Rinard. 2016. The challenges of staying together while moving fast: an exploratory study. In *Proceedings of the 38th International Conference on Software Engineering (ICSE'16)*. ACM. 982–993.

P. Runeson and M. Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Eng*. 14 (2009), 131–164.

G. Russell, F. Burns, and A. Yakovlev. 2012. VARMA—VARiability modelling and analysis tool. In *Proceedings of the IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS'12)*. IEEE. 378–383.

H. Samih and R. Bogusch. 2014. MPLM - MaTeLo product line manager: [relating variability modelling and model-based testing]. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*. ACM 127–142.

I. de Sousa Santos, R. M. C. Andrade, and P. de Alcântara dos Santos Neto. 2015. Templates for textual use cases of software product lines: results from a systematic mapping study and a controlled experiment. *J. Software Eng. R&D* 3 (2015), 5.

SAP Configurator. 2016. Retrieved from www.sap.com.

K. Schmid and M. Schank. 2000. PuLSE-BEAT — A decision support tool for scoping product Lines. In *Proceedings of the International Workshop Software Architectures for Product Families (IW-SAPF-3)*. Springer, Berlin, 65–75.

C. B. Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Trans. Software Eng*. 25, 4 (1999). 557–572.

S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés. 2012. BeTTy: Benchmarking and Testing on the Automated Analysis of Feature Models. In *Proceedings of the 6th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'12)*. ACM. 63–71.

D. Sellier, M. Mannion, and J. X. Mansell. 2008. Managing requirements inter-dependency for software product line derivation. *Requirements Engineering* 13, 4 (2008), 299–313.

P. Shakari and B. Møller-Pedersen. 2006. On the implementation of a tool for feature modeling with a base model twist. In *Proceedings of the Norwegian Informatics Conference (NIK'06)*. 81–93.

S. Sepúlveda, A. Cravero, and C. Cachero. 2016. Requirements modeling languages for software product lines: A systematic literature review. *Inf. Softw. Technol*. 69 (2016), 16–36.

S. She. 2016. Linux Variability Analysis Tools (LVAT). Retrieved from https://code.google.com/archive/p/linux-variability-analysis-tools/.

I. F. da Silva, P. A. da Mota Silveira Neto, P. O'Leary, E. S. de Almeida, and S. R. de Lemos Meira. 2011. Agile software product lines: A systematic mapping study. *Softw. Pract. Exper.* 41, 8 (2011), 899–920.

J. Sincero and W. Schröder-Preikschat. 2008. The Linux kernel configurator as a feature modeling tool. In *Proceedings of the 12th International Conference on Software Product Lines (SPLC'08)*. IEEE. 257–260.

M. Sinnema, S. Deelstra, J. Nijhuis, and J. Bosch. 2004. COVAMOF: A framework for modeling variability in software product families. In *Proceedings of the 3rd International Conference on Software Product Lines (SPLC'04)*. Springer, Berlin. 197–213.

M. Sinnema and S. Deelstra. 2007. Classifying variability modeling techniques. *Inf. Softw. Technol.* 49, 7 (2007), 717–739.

L. R. Soares, P. Potena, I. do Carmo Machado, I. Crnkovic, and E. S. de Almeida. 2014. Analysis of non-functional properties in software product lines: A systematic review. In *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications*. IEEE. 328–335.

SparxSystems Enterprise Architect. 2016. Retrieved from www.sparxsystems.com.

M. Stengel, M. Frisch, S. Apel, J. Feigenspan, C. Kästner, and R. Dachselt. 2011. View infinity: a zoomable interface for feature-oriented software development. In *Proceedings of the 33rd International Conference on Software Engineering (ICSE'11)*. IEEE. 1031–1033.

G. Succi, W. Pedrycz, J. Yip, and I. Kaytazov. 2001. Intelligent design of product lines in Holmes. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering*. IEEE. 75–80.

T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. 2012. FeatureIDE: An extensible framework for feature-oriented software development. *Sci. Comput. Program.* 79(2014), 70–85.

P. Trinidad, D. Benavides, A. Ruiz-Cortes, and S. Segura. 2008. FAMA Framework. In *Proceedings of the 12th International Software Product Line Conference (SPLC'08)*. IEEE. 359–359.

G. Vale, E. Figueiredo, R. Abílio, and H. A. X. Costa. 2014. Bad smells in software product lines: A systematic review. In *Proceedings of the 8th Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS'14)*. IEEE. 84–94.

J. White, D. C. Schmidt, E. Wuchner, and A. Nechypurenko. 2007. Automating product-line variant selection for mobile devices. In *Proceedings of the 11th International Software Product Line Conference (SPLC'07)*. IEEE. 129–140.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, and B. Regnell. 2012. Experimentation in Software Engineering. Springer, Berlin.

A. E. E. Yamany, M. Shaheen, and A. S. Sayyad. 2014. OPTI-SELECT: An interactive tool for user-in-the-loop feature selection in software product lines. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools*. ACM. 126–129.

H. Zhang and S. Jarzabek. 2004. XVCL: a mechanism for handling variants in software product lines. *Science of Computer Programming* 53, 3 (2004), 381–407.

B. Zhang and M. Becker. 2014. Variability code analysis using the VITAL tool. In *Proceedings of the 6th International Workshop on Feature-Oriented Software Development (FOSD'14)*. ACM. 17–22.

ZIPC Feature. 2009. Retrieved from http://www.zipc.com/english/product/xmodelink/.

Received March 2014; revised November 2016; accepted January 2017