Journal of Universal Computer Science, vol. 26, no. 10 (2020), 1282-1311
submitted: 26/2/20, accepted: 27/10/20, appeared: 28/10/20

Have Variability Tools Fulfilled the Needs of the Software Industry?

Ana Paula Allian
(State University of Maringá and
University of São Paulo
Maringá, Brazil
ana.allian@usp.br)

Edson OliveiraJr
(State University of Maringá
Maringá, Brazil
edson@din.uem.br)

Rafael Capilla
(Rey Juan Carlos University
Madrid, Spain
rafael.capilla@urjc.es)

Elisa Yumi Nakagawa
(University of São Paulo
São Carlos, Brazil
elisa@icmc.usp.br)

Abstract: For nearly 30 years, industry and researchers have proposed many software variability tools to cope with the complexity of modeling variability in software development, followed by a number of publications on variability techniques built upon theoretical foundations. After more than 25 years of the practice of software variability, there are not many studies investigating the impact of software variability tools in the industry and the perception of practitioners. For this reason, we investigate in this research work how existing software variability tools fulfill the needs of companies demanding this kind of tool support. We conducted a survey with practitioners from companies in eight different countries in order to analyze the missing capabilities of software variability management tools and we compared the results of the survey with the scientific literature through a systematic mapping study (SMS) to analyze if the proposed solutions cover the needs required by practitioners. Our major findings indicate that many tools lack important qualities such as interoperability, collaborative work, code generation, scalability, impact analysis, and test; while the results from the SMS showed these such capabilities are, to some extent, found in some of the existing tools.

Key Words: Software Variability Tools, Systematic Mapping Study, Interoperability, Scalability

Category: D.2, D.2.2, D.2.11, D.2.13

1 Introduction

For almost three decades software variability has been proven as the major software technique to describe the commonalities and variabilities of software systems and enabling the adoption of Software Product Lines for an effective mass-customization of products and tailored for different market segments [ISO/IEC, 2013a]. Variability is a significant ability of almost all software systems and not just related to the software product line (SPL) [Hilliard, 2010]. Due to its importance, variability should be managed properly throughout the software life cycle [Pohl et al., 2005], changing the development process, from requirements to implementation [Sierszecki et al., 2014]. The International Organization for Standardization (ISO) established ISO/IEC 26550, which encompasses a set of activities to create and maintain variants in all software development phases [ISO/IEC, 2013a]. Besides this, another standard, ISO/IEC 26555 [ISO/IEC, 2013b], establishes that variability management [Capilla et al., 2013] shall support variability models, variability binding times, variability documentation, variability tracing, and variability control and evolution. More importantly, software variability tools facilitating the modeling and maintenance of large variability models is an important concern, especially in industrial settings [ISO/IEC, 2012].

To date, we have witnessed the evolution of a significant variety of research software variability tools exhibiting different capabilities and according to the evolution of software variability modeling approaches. Also, a couple of commercial tools (i.e. GEARS and pure::variants) are preferred by industry to launch a product line approach. Many of the research, open-source, and commercial tools exhibit different capabilities (e.g., automatic code generation and traceability) but there are important lacks and concerns not properly addressed by one single tool. First, interoperability between different data formats supporting the variability models and to other software artifacts like software requirements is an important concern for the plethora of existing software variability tools. Second, many tools lack proper visualization capabilities able to display properly large variability models. Third, almost none of the tools are prepared to support runtime variability concerns, maybe because customers still don't demand this kind of facilities and systems are reconfigured at post-deployment time using a different approach. Thus, the desired scalability to have open variability models changing variants at any time is the capability to be supported in the future. Fourth, modeling context variability and its integration with context-oriented programming languages are non-existent. Finally, code generation once variability models are configured is a capability provided only by some tools.

The main contribution of this work is to present both the state of the practice and the state of the art on software variability tools and main capabilities still required by industry. To find the state of the practice, we firstly identified the tools known by practitioners from industry, their use, and the capabilities still uncovered by such tools, by conducting a survey widely distributed in many companies, from which we had 28 respondents from eight different countries. Additionally, we conducted a systematic mapping study (SMS) aiming at analyzing whether the capabilities required by practitioners have been already solved by solutions found in the state of the art identified in the literature. As the main result, the most missing capabilities in tools used in industry are interoperability, collaborative work, code generation, scalability, impact analysis, and test, while the SMS revealed no single tool offers all desired capabilities required by practitioners and existing tools are not also prepared to support the development of large, complex software-intensive systems in iterative, distributed software development processes.

This article is structured as follows: Section 2 describes some related works to this research. In Section 3 we outline the research method to conduct this study; Section 4 presents results of the survey; Section 5 discusses results of the SMS; Section 6 provides the discussion putting together results of both survey and SMS; Section 7 presents the threat to validity of our work; and, finally, Section 8 concludes this work.

2 Related Works

Several research works have already investigated software variability tools. [Lisboa et al., 2010] performed a systematic literature review (SLR) on *domain analysis tools including variability tools*, and identified whether such tools could document common and variability features of software systems; besides, important functionalities were identified, such as planning (to collect data and identify domain features based on scope definition), modeling (to represent and model mandatory and variable features), and validation (to validate consistencies in variability models). Following, [Pereira et al., 2015] conducted an SLR on *SPL tools and variability tools* and classified them into the same classification previously found by [Lisboa et al., 2010]. Moreover, [Benavides et al., 2010] performed a comprehensive literature review on *tools to the automated analysis of feature models*, covering only some tools identified in our study. These three studies brought an important panorama on variability tools, but none addressed the use of these tools in the industry.

[Berger et al., 2013, Berger et al., 2014] reported results of interviews with industry practitioners about the use of variability tools (including variability modeling), their benefits and limitations, but limited to the use in the SPL context, while some variability tools are missing in their analysis. [Bhumula, 2013] performed an online survey to identify variability tools in use and carried out a comparative analysis on 14 tools organizing them into different criteria (governance issues and technical aspects like variability approach, visualization of feature models, binding time, and feature types). This author recommended a set of best tools for industry but did not consider a complete list of tools and also the results have been published six years ago.

In a more recent study, [Bashroush et al., 2017a] identified using a systematic literature review (SLR) a comprehensive list of variability tools used in the context of software product lines. Complemented by a survey with industry practitioners the authors look for the five most important quality attributes of SPL tools, such as performance, scalability, integration, and usability. As a result, the authors found that most tools did not mention the quality attributes pointed out by practitioners. Although this study is closely related to our work, they didn't analyze major capabilities in the tools missed by practitioners. Compared to our work, we cover more tools than [Bashroush et al., 2017a] and specific to software variability management, not to support of the software product line engineering practice. Also, instead of analyzing the adoption of the tools by industry, we investigated the exiting and missing capabilities of the tools providing a complimentary analysis to previous works.

Other related works, like [Mahdavi-Hezavehi et al., 2013], provide a systematic literature review about variability in quality attributes of service-based systems but the study describes only the qualitative part and find evidence of methods for handling variability in quality attributes which can be applicable to design-time and run-time. In [Raatikainen et al., 2019] the authors describe a tertiary study on software product lines and variability modeling but the authors focus only on the variability modeling part neglecting other dimensions supported by current variability management tools but also they don't provide any discussion about such tools. Finally, a similar study to ours described in [Bashroush et al., 2017b], the authors perform a systematic literature review of 37 software variability management tools in order to understand the tools' characteristics and their maturity. Although the analyzed the main characteristics of the tools, the final selection to compare these tools is based on their ability to support a product line approach and not specific for variability management only. Therefore, although we share some comment goals and results regarding the important features identified, our work provides some additional results not found in this work.

3 Research Method

Our research method involved three main activities: (i) conduction of a survey; (ii) conduction of an SMS; (iii) mapping of capabilities in variability tools required by industry with capabilities found in the SMS. Details on the design and execution of both survey and SMS are presented in this section (Subsections 3.1 and 3.2), while analysis of the results from the survey and SMS are discussed in Sections 4 and 5, respectively. Finally, the last activity (i.e., the mapping) is described in Section 6. Figure 1 illustrates our research method[1].

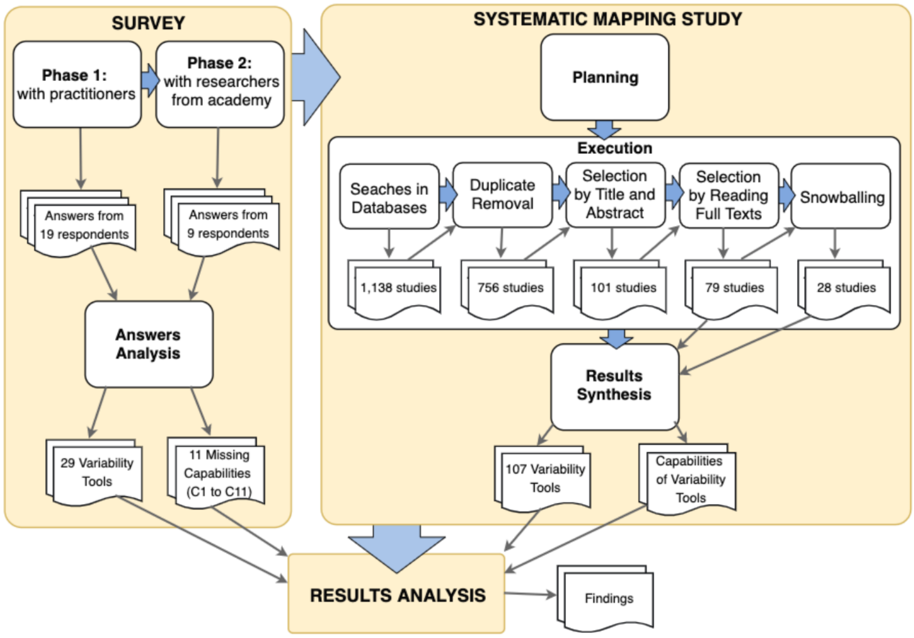

Figure 1: Research method used to conduct this work

3.1 Survey Design and Execution

Nowadays, surveys are widely applied to gather human opinions on different aspects via questionnaires or interviews [Wohlin et al., 2012]; hence, we conducted a survey with practitioners and/or users of variability tools to collect evidence about the real use of such tools in the industry. We strictly followed the three-step process for surveys defined in [Wohlin et al., 2012]: survey design, execution, and analysis of results. With regard to the survey design, we concerned about its protocol to ensure rigor and repeatability of this study. Specifically, this protocol established three questions (Q)[2]:

[1] All data from the survey and the SMS is available at `https://doi.org/10.5281/zenodo.3406370`

[2] We adopted Q when referring to the questions of our survey and RQ to the research questions of our SMS intending to avoid misunderstanding.

– *Q1: Which software variability tool(s) have you used?*

  Rationale: We would like to find variability tools used by practitioners in software projects.

– *Q2: How easy is it to use software variability tool(s)?*

  Rationale: We would like to know if existing tools are easy to be used from the practitioners' perspective.

– *Q3: Which capabilities do you miss in software variability tools and which others could be improved?*

  Rationale: We would like to identify the main capabilities still required by industry in variability tools.

We adopted a self-administered, cross-sectional, and exploratory survey according to [Molléri et al., 2016]. By being self-administered, respondents could answer in writing a set of questions; by being cross-sectional, we could gather a snapshot in time, as this survey could give us an idea on how things are for our respondents; finally, by being exploratory, we could focus on taking advantage of the respondent's experience to identify capabilities of software variability tools.

To reach wider dissemination of this survey, we designed the questionnaire using an online survey[3]. Before distributing this questionnaire, it was systematically evaluated by two experienced researchers on software variability.

Regarding the survey execution, we divided it into two phases. In *Phase 1*, we invited by e-mail 44 practitioners from SME and large companies and people involved in past SPL Hall of Fame[4] projects. In addition, we asked practitioners to forward the questionnaire or suggest other potential respondents. By sending a personal invitation, in some cases, emails were exchanged to explain the intention of the survey. We got 19 responses but unfortunately none from HoF projects. However, in order to increase the confidence of the responses, we only considered subjects with at least three years of experience using software variability tools. In *Phase 2*, we extended the survey to researchers from academia with an experience similar to the industry practitioners in the development, evaluation, and use of variability tools. We invited 25 researchers, 9 of them answered the questionnaire. Finally, we had a total of 28 respondents from 8 different countries.

3.2 SMS Planning and Execution

To systematize the SMS conduction, we followed the SMS guidelines [Petersen et al., 2015, Kitchenham and Charters, 2007], and herein we present the main elements of this protocol that was rigorously established to reduce the possibility

[3] LimeSurvey, available in https://www.limesurvey.org.
[4] http://splc.net/hall-of-fame/

of biases. First of all, considering that the specific objectives of this SMS were to identify existing variability tools or prototypes found in the literature and their capabilities, we established the following research questions (RQs):

- *RQ1: Which software variability tools (even as a prototype) have been published in the literature?*
- *RQ2: Which are the main capabilities of these variability tools?*

Regarding the search strategy, which supports the identification and retrieval of as many relevant studies as possible, a search string was defined using terms that entailed the most appropriate keywords pertaining to the scope of this SMS:

(("variability management" OR "variation point" OR "feature model") AND (SPL OR "product line" OR "product-line" OR "product family" OR "family of products" OR "systems family" OR "family of systems") AND (tool*))*

Following, primary studies were collected from February to April, 2019 from the most important electronic databases for computing: ACM Digital Library (ACM DL), IEEE Xplore Digital Library, Google Scholar, Ei Compendex, Science Direct, and Scopus. The search string was used against the metadata (titles, keywords, and/or abstracts) of selected databases. In particular, as Google Scholar does not support complex strings, the search was made using the advanced Google tool by combining the keyword according to search operators[5]. Because of the variety in the search mechanisms of these databases, we run different search queries, totaling 1,138 studies recovered (112 from ACM DL, 107 from IEEE Xplore Digital Library, 109 from Google Scholar, 236 from Ei Compendex, 97 from Science Direct, and 477 from Scopus). Studies were imported into the Mendeley tool and 382 duplicated studies were automatically removed, remaining 756 unique ones.

For the selection of studies, we defined one inclusion criterion (IC) and two exclusion criteria (EC):

- IC1: The study is a peer-reviewed scientific publication that addresses software variability tool(s).
- EC1: The study is related to software variability tools without describing capabilities or availing any documentation.
- EC2: The study presents the same tool already presented in a more complete study.

[5] Advanced Operators for Google Search`http://www.googleguide.com/advanced_operators_reference.html`

- EC3: The study is a short paper, editorial, keynote, opinion, tutorial, poster or panel.
- EC4: The study is not written in English.
- EC5: The study is a non-peer-reviewed publication.
- EC6: The full text of the study is not available.

We applied these criteria in two rounds of selection. In the first one, the first author of this paper read the title and abstract of each study and, in cases where they were not enough to decide whether the study should be included or excluded, the introduction and conclusion sections were also read. All borderline papers were reviewed by the other three authors to double-check the fit under the inclusion criteria. As a result, we removed 655 studies and selected 101. In the second round, the full text of all selected studies was read by the four authors to find out whether they meet the IC/EC criteria. All conflicting cases were discussed by the four authors. When a study did not provide enough information, we also looked for other related studies or external material describing the tool (e.g., website and technical reports). This activity ended with a selection of 79 studies. It is worth highlighting in both rounds, we selected the more complete study related to a given tool; therefore, we had identified 79 tools.

Furthermore, each of the 79 studies was carefully reviewed by applying the snowballing technique [Wohlin, 2014, Wohlin et al., 2012]. All references from each study were examined in the *first backward snowballing iteration* and 25 new studies were included. Google Scholar was used in the *first forward snowballing iteration* to verify citation from each study and 3 new studies were included. A second snowballing iteration (backward and forward) was performed and no new studies were found. Therefore, 28 new studies were added through snowballing, totaling 107 studies (i.e., 107 tools) for further analysis.

4 Analysis of the Survey Results

We analyzed answers of 28 respondents, who took on average 8.2 minutes to answer the questionnaire, which can possibly indicate no fatigue effect. Besides identifying the respondents' profile, we collected the list of tools, the usability level, and the missing capabilities from the respondents' perspective, as well as open issues related to variability tools.

With regard to the roles and experience of the respondents in software projects that have used variability tools, all participants have at least 3 years of experience in the development, evaluation, and/or use of variability tools. Some participants from the industry have more than one role as senior software engineer (10 participants), software architect (8), software developer (4), enterprise architect (3), junior software engineer (2), business/technical/project manager (2), business analyst/requirements engineer (1), and tester (1). Most of the respondents (11 participants) have on average more than 10 years of experience in their roles, 8 participants declared an average from 5 to 9 years of experience, and 9 participants informed at least 4 years of experience.

Therefore, our respondents are well experienced in different roles in the software development process and even as researchers in the area, which can provide us some confidence in their answers regarding variability tools.

Regarding *Q1: Variability tools used by practitioners*, in general, participants have experience with more than one tool. 29 different tools were mentioned; the most cited ones were pure::variants (18 answers), followed by FeatureIDE (15), SPLOT (13), fmp (9), CLAFER (8), Gears (8), FAMA (7), CVL (5), Hephaestus (5), and CaptainFeature (3). Other 3 variability tools were mentioned twice: Genarch, PLUM, and VariaMos. Besides that, 16 tools were mentioned once: COVAMOF-VS, DecisionKing, Dopler, Hydra, Kumbang Tools, Lisa, Visit-FC, V-Manage, XFeature, PlugSPL, FW Profile, EASy-Producer, Kconfig, TypeChef, CVM, and PreeVision. We observed the most of these tools were developed by academia, what is a very interesting finding considering the direct impact of academic results in the industry, except only pure::variants, Gears, FW-Profile, PLUM, and Pree-vision that are from the industry; in particular, pure::variants provides a community version.

To *Q2: Ease to use software variability tools*, respondents selected a scale for each tool as: Very Easy, Easy, Moderate, Hard, and Very Hard. The top 10 tools (also mentioned in Q1) are listed in Table 1, together with their usability level according to the respondents; e.g., the 18 respondents with experience in pure::variants consider it Very Easy (for 3 respondents), Easy (4), Moderate (9), Hard (2), and none Very Hard. Hence, observing this and other tools, they seem to be easy or moderately easy to use, what is an important finding; however, as usability is a quite subjective matter, results can be considered users' impression.

With regard to *Q3: Capabilities missed in variability tools (or capabilities that should be improved)*, Figure 3 summarizes the 11 missing capabilities (C1 to C11) mentioned by respondents and the percentage of respondents who pointed them. Considering these capabilities are the core of our investigation, we decided to detail them below:

C1 - Interoperability/Integration: This capability represents the ability of software systems to interact among them and share information towards mutually common goals. Interoperability incorporates content from independent systems/tools to work together and exchange data among them in a collaborative way [Chituc, 2017]. In the same perspective, integration enables tools to work together as one solution, combining two or more components/systems into a larger system to satisfy specific objectives. Regarding

Table 1: Number of respondents that classified the top 10 variability tools according to the scale of usability

Tools	**Very Easy**	**Easy**	**Moderate**	**Hard**	**Very Hard**
pure::variants	3	4	9	2	
FeatureIDE	4	7	2	2	
SPLOT	7	4			
CLAFER	1		4	2	1
fmp	2	4	1		
Gears		1	4	2	
CVL Tool			2	2	1
FAMA		1	3		1
Hephaestus		1	2	1	
CaptainFeature	1	1	1		

variability tools, interoperability was the main missing capability. In general, these tools could better interoperate to IDE (Integrated Development Environments), cloud-based environments, testing tools, and version control, such as GIT[6] and Apache Subversion (SVN)[7]. Considering the top 10 tools (previously showed in Figure 1), commercial tools like pure::variants and Gears provide a set of integrated tools, such as object-oriented modeling tools, requirement management systems, and consistency checks analysis tools. Considering research and open-source tools, in particular, FeatureIDE, CLAFER, CVL Tool, and FAMA, they provide some integration mechanisms with modeling tools and consistency check tools. However, many tools should still include this capability if the intention is their sustainability along the time.

C2 - Code generation: This capability make it possible to automatically obtain source code (even as a general structure) from variability models, which could reduce errors and improve productivity. In particular, from our survey, a code generator is a relevant capability in variability tools. Checking the top 10 tools, pure::variants, Gears, FeatureIDE, and CVL Tool are the unique tools with this capability, but considering the universe of variability tools, many of them need still evolve to encompass it.

C3 - Impact analysis: This capability makes it possible to evaluate consequences (or potential effects) of future changes upon software systems. Regarding variability tools, addressing impact analysis is much more complex than expected, due to a higher degree of dependencies among artifacts,

[6] https://gitscm.com/
[7] https://subversion.apache.org/

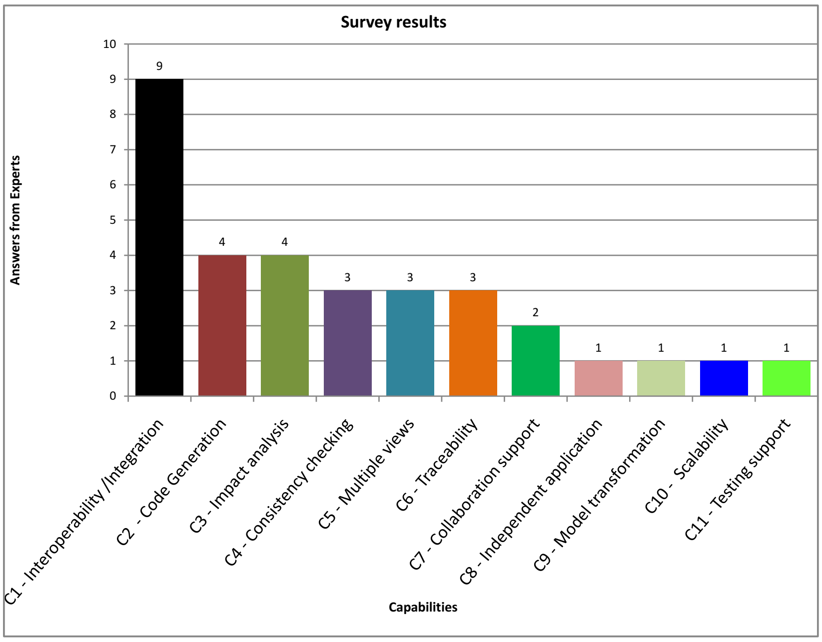

Figure 2: Missing capabilities in variability tools identified in the survey

which are affected by changes in requirements or decisions. For this reason, tools must be implemented with policies governing how changes could be reviewed and accepted by stakeholders.

C4 - Consistency checking: This capability makes it possible to evaluate variability models against pre-defined conformance rules during the configuration process. Results of our survey pointed out some tools presented in Table 1, including SAT Solvers, BDD, and CSP, have to support to consistency checking mechanisms; however, solvers and reasoners (that usually implement this capability) are known as NP-complete problems and can leverage the analysis of features to exponential worst-case running time [Pohl et al., 2013]. For this reason, a proper mechanism must be carefully selected to improve efficiency when checking variability models.

C5 - Multiple views: This capability makes it possible to describes different views of a system to support stakeholders to manage variability. Respondents informed that different views are still needed to address different stakeholders' concerns. Observing the top 10 tools, some of them combine graph visualization with a file tree and coding area.

C6 - Traceability: It is the capability to trace software artifacts both forward and backward along the software development process. Respondents declared the need for traceability support between feature models and requirements. In addition, they also required traceability from architectural models to implementation code.

C7 - Collaborative support: It can manage the agreement among parties to make possible their collaboration during software projects. Regarding variability tools, this capability could connect diverse, multidisciplinary teams and improve their communication; however, such tools in general lack of this capability;

C8 - Independent application: This capability makes variability tools be fully independent system instead of extensions or plugins of another platform/system. In this way, variability tools could be developed independently with, for instance, a graphical editor and would facilitate the creation of variability models.

C9 - Model transformation: It refers to an automated or semi-automated functionality in tools to modify and/or create software models. Then, variability tools could present this capability to support transformation between different model formats, besides mechanism to generate output in different formats, such as XML and XSL.

C10 - Scalability: It is the capability to handle large variability models with multiple dependencies among them. Scalability has been a great challenge when considering millions of features (and variants) in the variability model. Ways to improve such scalability and usability of tools concerning adequate model visualization are still a big challenge.

C11 - Integration with testing tool: This is the capacity to select features (or variants) to be tested as part of software systems. This capability could anticipate the finding of errors and, considering the top 10 tools, pure::variants and Gears are the unique ones that support the integration with test tools, which leverage the capacity of these variability tools; however, most tools do not provide this capability.

We observed there are many tools available and in use, but according to the survey results, practitioners still miss important capabilities, in particular, interoperability/integration with other development tools. Following, we present the results of our SMS on variability tools aiming to extract whether such tools have the capabilities required by practitioners.

5 Analysis of the SMS Results

As a result of the SMS execution, we found 107 tools[8] listed in chronological order, together with the tool's name (Column Tool), publication year[9] (Year, also a link to the reference of the study), capabilities (C1 to C11, further detailed in RQ2), if developed (Dev) by academia (A) or industry (I), and if available for download (Av). In addition, we summarize in Tables 2 and 3 which of the tools analyzed support each of the eleven capabilities discussed below.

With regard to *RQ1: Variability tools published in the literature*, we found this expressive number of tools developed in the last 25 years. Most of them were developed by academia (100 tools), while 7 tools by industry: pure::system, Gears, MetaEdit+, PLUM, 001, DOORS, and v.control. All tools mentioned by practitioners in the survey (Section 4) were also found in our SMS, except three: (i)Captain Feature is a feature modeling tool for modeling features and encompasses an integrated configurator to create specialized feature diagrams of software system families; (ii) PREEvision is a model-based development tool for embedded systems for the automotive domain that enables developers to model variant features according to AUTOSAR Adaptive Platform[10]; (iii) FW Profile makes it possible to use a specific modeling language to model the behavior of software applications separating functional and non-functional behavior through variant properties. No publication is available, but only technical reports on its website.

Regarding *RQ2: Main capabilities of variability tools*, Tables 2 and 3 (columns C1 to C11) presents the capabilities extracted from each tool/study (cell marked with a dot). It is worth saying that during the analysis and extraction of information from each study, some studies did not provide sufficient data and/or information was clearly described, then we decided to keep the cell empty. Figure 3 summarizes the number of tools that have each of the 11 capabilities, e.g., 76 of 107 tools (i.e., 71%) provide C4 (consistency checking). To better understand the distribution of capabilities developed during the period of a year by each tool, we designed a bubble plot, as presented in Figure 4. This presents the frequency of tools developed by a period of years. Capabilities such as *C1 - Interoperability /Integration, C2 - Code Generation, C4 - Consistency checking, C9 - Model transformation, and C11 - Testing support* have gain more attention during years 2006 to 2015. This can be explained by the transformation of Industry into the fourth industrial revolution, which demands new ways to interconnect systems, create models, and evaluate the products in favor to increase the production line [Ihme et al., 2014]. Following, we discuss in details on each

[8] `https://doi.org/10.5281/zenodo.3688405`

[9] The year might not be the same one of the tool's release, as we selected the more complete/relevant study related to that tool.

[10] https://www.autosar.org/

capability:

C1 - Interoperability/Integration: 56% of tools (i.e., 60 tools) have integration/interoperability capability. We can highlight three tools that most present such capability: DOORS, Gears, and pure::variants. DOORS implements two interoperability options for DOORS Next Generation, which supports users to exchange data between IBM applications. pure::variants and Gears have a strong focus on interoperability and extensibility with many tools, such as Eclipse Modeling Framework (EMF), object-oriented modeling tools, code generators, configuration management systems, UML and SDL descriptors, and many other tools to support stakeholders during the software development. These two tools interoperate with, for instance, AUTOSAR, Simulink[11], UML/SysML modelers, among others, providing support to the entire software life cycle. Moreover, many other tools, including Feature Plugin, Wecotin, fmp, PLUSS Toolkit, FAMA, FeatureMapper, CVL tool, FMT, XToF, metadoc, VariaMos, CLAFER, Hydra, and FeatureIDE, also integrate with EMF to generate tree-oriented models and with reasoners and Boolean solvers to support consistency analysis of feature models. The integration with EMF is important because it is one of the most disseminated solutions to model features.

C2 - Code generation: Many tools (46.7% of studies, i.e., 50 tools) support this capability due to the integration with EMF, which provides mechanisms to support code generation to produce Java classes from the variability metamodel. Examples of tools with EMF code generation are pure:variants, Feature Plugin, BVR tool, fmp, PLUSS Toolkit, FAMA, FeatureMapper, CVL tool, FMT, metadoc, Kumbang, CLAFER, Hydra, and FeatureIDE. Differently from these tools, we can highlight three tools that provide other means: (i) ASADAL generates code by processing macros embedded in various design models and components; (ii) GenArch uses code annotation to indicate implementation of features and in the code; and (iii) MetaEdit+ uses a layer to generate code with standard libraries to fit into a specific domain.

C3 - Impact analysis: This capability is related to policies addressing how changes in the system requirements are managed and accepted to generate new products. 23.4% of tools (25 tools) support this capability. This is the case of tools like Gears, COVAMOF-VS, Metadoc, FAMILIAR, VariaMos, BETTY, VULCAN, LISA, CLAFER, Odyssey, ToolDAy, Doors, DarwinSPL, FORCE, Holmes, ASADAL, and also pure::variants. In particular, we can highlight how the impact analysis is performed by some of these tools. Gears reduces the number of options during the impact analysis and sim-

[11] https://www.pure-systems.com/products/pure-variants-for-simulink-288.html

Table 2: Capabilities of Software Variability tools

Nr.	Tool	Ref	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	Dev	Av
1	001	001/Hamilton	•	•	•	•	•	•			•		•	I	Yes
2	Domain DSSA	DssaDomain/Tracz		•										A	No
3	DARE-COTS	Dare/FrakesPF97												A	No
4	Odyssey	Odyssey99	•		•		•	•		•	•			A	Yes
5	SSEP toolset	SSEP/StuartSPCWC00	•			•		•						A	No
6	Hollmes	Holmes/SucciYP01	•		•		•							A	No
7	XVCL	XVCL02		•										A	No
8	V-Menage	Menage/Mansell					•			•				A	Yes
9	Wecotin	Wecotin03	•			•	•	•						A	Yes
10	V-Menage	Menage/Mansell					•			•				A	Yes
11	AHEAD	Ahead/BatorySR04		•		•						•		A	Yes
12	DREAM	Dream04						•						A	No
13	pure::Variants	Purevariants/BeuchePS04	•	•	•	•	•	•	•		•	•	•	I	Yes
14	RequiLine	Requiline/MassenL03				•	•			•		•		A	No
15	FeatureIDE	FeatureIDE/Khan	•	•	•	•	•		•		•			A	Yes
16	FeaturePlugin	FMP/CzarneckiAKLP05a	•	•		•	•							A	Yes
17	fmp	FMP/CzarneckiAKLP05a	•			•	•							A	Yes
18	PLUSS Toolkit	Pluss05	•			•	•	•						A	No
19	XFeature	Xfeature05		•		•					•			A	Yes
20	ASADAL	Asadal/KimKASCK06		•	•		•			•			•	A	No
21	Covamof-VS	Covamof/Sinnema			•		•						•	A	No
22	REMAP	Remap/SchmidKE06	•					•						A	No
23	YaM	Yam06	•	•							•		•	A	No
24	DECIMAL	Decimal/Dehlinger	•	•		•	•			•				A	No
25	DecisionKing	DecisionKing07	•				•	•			•			A	No
26	FAMA	Fama/Roos-FrantzGBC12	•	•		•	•							A	Yes
27	GenArch	Genarch/CiriloKL07	•	•		•					•		•	A	No
28	Kumbang	Kumbang/MyllarniemiRM07				•	•							A	Yes
29	MetaEdit+	Metaedit/Tolvanen07a		•			•			•	•			I	Yes
30	VMWT	VMWT2007				•				•				A	No
31	Cide	Cide08	•	•		•								A	Yes
32	Easy-Producer	EasyProducer/EichelbergerS08	•	•		•	•							A	Yes
33	FeatureMapper	FeatureMapper08	•				•				•			A	No
34	Flip	Flip/SoaresCNMBA08	•	•							•			A	No
35	Gears	Gears/Krueger08	•	•	•	•	•	•	•	•	•	•	•	I	Yes
36	Genie	Genie08		•		•	•	•			•			A	No
37	LKC	LKC08		•		•								A	Yes
38	MoSPL	Mospl08		•		•								A	No
39	PLUM	Plum/LopezM13		•		•	•				•			I	No
40	Visit-FC	Visitfc/BotterweckTCNP08						•						A	No
41	CVL tool	CVL/Haugen	•	•		•	•				•			A	Yes
42	CVM	CVM10		•		•	•						•	A	Yes
43	Hephaestus	Hephaestus/TurnesBAL11		•		•				•	•			A	Yes
44	Hydra	Hydra09	•			•								A	Yes
45	S2T2	S2t2/Botterweck	•	•		•					•			A	Yes
46	SPLOT	SPLOT/Mendonca				•				•				A	Yes
47	CDL	CDLKconfig10	•	•		•					•		•	A	Yes
48	FMT	FMT/Laguna10	•	•				•			•			A	No
49	Kconfig	CDLKconfig10	•	•		•					•		•	A	Yes
50	MUSA	Musa/Bashroush10								•		•		A	No
51	TypeChef	TypeChef10				•								A	Yes
52	XToF	Xtof/GauthierCBHSM10	•	•				•						A	No
53	DOPLER	Dopler/DhunganaGR11	•	•		•								A	Yes

Table 3: Capabilities of Software Variability tools (continuation)

Nr.	Tool	Ref	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	Dev	Av
54	metadoc	Metadoc/ThurimellaJ11	•	•	•	•		•				•		A	No
55	FAMILIAR	FAMILIAR11	•	•	•	•						•		A	Yes
56	MoSo-PoLiTe	Mospolite/OsterZML11	•										•	A	No
57	SPLVerifier	SplVerifier/Apel				•								A	Yes
58	ToolDAy	Toolday/LisboaGAM11	•		•	•	•	•						A	No
59	View Infinity	ViewInfinity11		•			•	•				•		A	Yes
60	ABS tool	Abs/WongAMPSS12	•	•	•	•						•	•	A	No
61	BeTTy	BeTTy12	•	•	•	•	•						•	A	Yes
62	DPLfw	DPLfw/GomezPCBL12					•							A	No
63	EPM	EPM/AbeleLRWG12		•		•	•							A	No
64	ISMT4SPL	Ismt4spl/ParkRB12	•	•		•	•	•		•				A	No
65	LISA Toolkit	Lisa/GroherW12	•	•	•	•	•	•			•			A	No
66	Moskitt4SPL	Moskitt4SPL2012	•		•						•			A	No
67	PlugSPL	PLUGSPL		•										A	No
68	SNIP	Snip/classen2012model	•			•		•			•			A	No
69	Sysiphus	IVMM/ThurimellaB12	•			•		•						A	No
70	VariaMos	Variamos/MazoMRST12	•		•	•	•				•	•		A	Yes
71	VARMA	Varma12			•	•				•				A	No
72	VULCAN	Vulcan/LeeYK12	•	•	•	•							•	A	No
73	CLAFER	Clafer/AntkiewiczBMOLC13	•	•	•	•	•			•				A	Yes
74	DOORS	Doors13	•		•		•	•	•	•				I	Yes
75	Invar	Invar/DhunganaSBRGBG13	•			•				•				A	No
76	SOASPL	Soaspl/Abu-MatarG13		•		•	•				•			A	No
77	Matelo	Matelo/SamihB14	•			•		•					•	A	No
78	MPLM-MaTeLo	Matelo/SamihB14		•									•	A	Yes
79	OPTI-SELECT	OPTISELECT14			•	•	•			•				A	No
80	SPL Config	Splconfig		•		•	•							A	Yes
81	Varies	Varies/WagnerDHTGK14	•			•		•			•			A	No
82	VITAL tool	Vital14	•		•	•				•				A	No
83	ViViD	Vivid14		•		•								A	Yes
84	VMC	VMC14			•	•								A	No
85	WebFML	WebFML/BecanNAB14				•								A	Yes
86	BVR Tool	BVR/VasilevskiyHCJS15	•			•		•					•	A	Yes
87	CMT / FDE	CMTFDE15												A	Yes
88	SBAT	Sbat15	•			•								A	No
89	SPLicing	SPLicingTabasco15	•			•								A	No
90	SPL-TuPI	SPL-TuPI					•							A	No
91	SuperMod	SuperMod15	•			•	•							A	Yes
92	UC2FM	UC2FM15				•								A	No
93	v.control	Vcontrol15	•			•	•			•				I	Yes
94	XMAN	Xman15		•			•						•	A	No
95	Archfeature	ArchFeature16	•					•						A	No
96	CardyGAn	CardyGAn	•			•								A	Yes
97	RiPLE-HC	RiPLEH16		•		•								A	Yes
98	SVL tool	SVLTool				•								A	No
99	Zen-Config	ZenConfigurator16				•						•		A	Yes
100	DarwinSPL	DarwinSPL17	•		•	•	•	•			•			A	No
101	FixOnto	FixOnto17						•			•			A	No
102	FLAME	Flame2017				•				•			•	A	Yes
103	FORMAT	Format17				•							•	A	Yes
104	MOPPET	Moppet17	•	•		•					•			A	No
105	FORCE	Force2018	•		•	•		•				•		A	No
106	MIC	MIC18	•			•		•			•			A	No
107	Varion	Varion18	•			•		•						A	Yes

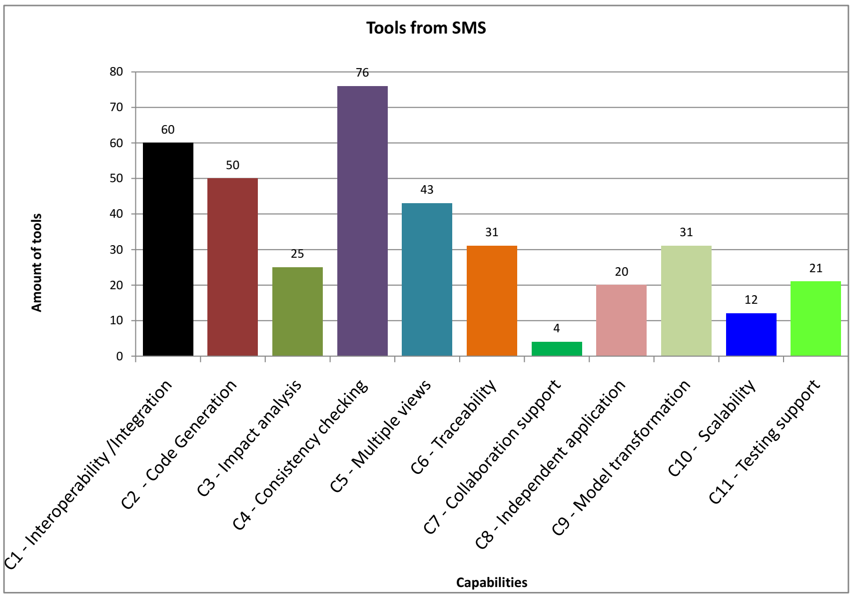

Figure 3: Capabilities available in software variability tools identified in the SMS

plifies the choices when defining an SPL approach. COVAMOF-VS automatically infers choices on a higher level of abstraction to reduce human effort to configure products. Metadoc provides deficit analysis to create consistent specifications of products. FAMILIAR uses the impact analysis to automatically propagate software product choices. VariaMos analyzes all valid products of an SPL and provides decision propagation. BETTY evaluates the performance of analysis in both average and pessimistic cases. VULCAN supports the analysis of source code, recovering a feature model. LISA supports continuous synchronization of an architecture to keep it up-to-date. CLAFER supports trade-off analysis from pre-configured variants versus optimal variants. Therefore, it is observed tools have adopted different strategies to deal with impact analysis. Besides that, to facilitate the impact analysis, tools also need to support traceability from requirements to code; however, both capabilities (impact analysis and traceability) were described only in studies related to 10 tools: pure::Variants, 001, Gears, Metadoc, LISA, Odyssey, ToolDAy, Doors, DarwinSPL, and FORCE.

C4 - Consistency checking: This capability is supported by most tools (71% or 76 tools). In general, variability tools integrate with consistency checking tools that provide logical solvers or arithmetic verifiers (i.e., propositional logic with SAT solvers, BDD, CSP, and Descriptive Logic). From the analysis of SMS results, we conclude consistency checking is maturing with an increasing number of contributions.

C5 - Multiple views: 40.2% of tools (43 tools) present more than one type of view to support stakeholders during the variability management. These views are generally split into two different aspects: problem space and solution space. The former is managed with graphs through a feature model-based approach or UML diagrams, while the latter can be represented by variability mechanism in code-based assets or components and their interactions, interfaces, and nodes. Furthermore, commercial tools like pure::variants and Gears also consider different viewpoints to support requirement engineers, domain analysts, developers, and software architects, due to the fact they integrate to many different software development ecosystems.

C6 - Traceability: 29% of tools (31 tools) support traceability between problem space (i.e., requirements, feature models, and architecture models) and solution space (i.e., code, components, and interfaces). Traceability helps stakeholders to understand the impact that a variation has at the decision-making process as it maps the domain space to the solution space. Traceability can still be considered a challenge, as many tools cannot still explicitly perform this capability.

C7 - Collaborative support: We found only 4 tools (3.7% of tools) describing some means to improve collaboration among their stakeholders. This is the case of Gears, which provides an efficient means to create and evolve its portfolio of features and requirements with a concise communication between business and engineering teams. DOORS makes possible global teams to work together during the requirements management in software projects. FeatureIDE provides a collaboration diagram to support communication among different stakeholders. pure::variants, in its enterprise edition, provides online collaborative support enabling organizations to create documents and models. Although important, this capability is also a challenge if variability tools will need to achieve another level of completeness.

C8 - Independent application: 18.7% of tools (20 tools) are described as an independent systems, while the others are dependent of other systems/platforms and most of them were implemented in Java as plugins.

C9 - Model transformation: 29% of tools (31 tools) have this capability that helps stakeholders to transform variability models into desired outputs, which could be source code or another model. Most of these tools are supported by EMF to interpret models and create a product variant or configuration files. This is the case of pure::variants, FeatureMapper, FeatureIDE, S2T2, Odyssey, CVL, LISA Toolkit, GenArch, XFeature, and PLUM. Other tools, such as PACOGEN, Varies, and VariaMos, use this capability to generate constraint models for configuring proper products. Model transformation in fact used to support stakeholders during the transition from problem space to solution space; hence, it can automate many of the steps to generate software products.

C10 - Scalability: Few studies (11.2% of studies or 12 tools) can deal with scalability of variability models. MUSA adopts hyperbolic trees to represent variability that can display more than one thousand features in a mind mapping visualization technique. AHEAD uses algebraic models to synthesize the scalability of models. View Infinity supports scalability with different feature model layouts and views. FAMILIAR supports the scalability of models by separating them into different concerns with automated reasoning support. Results from SMS indicate scalability is a key challenge of variability management tool, as it has been rarely described in the studies.

C11 - Integration with testing tool: 19.6% of tools (21 tools) support this capability. Tools like pure::variants, Gears, ABS, and GenArch use unit test cases to test the correctness of the implementation. In the same perspective, Matelo and MPLM-Matelo extends model-based testing formalism to generate test cases for each variant. Tools that make it possible to test the feature models are YAM, FORMAT, BeTTy, and Pacogen. MoSo-PoLiTe supports the pairwise test by combining a feature model and a test model, which is created for a selected configuration.

ASADAL provides methods to model a virtual test environment through UML Statecharts and a simulation-based environment. Finally, this capability in variability tools can ensure more reliability of feature models and products as well; however, this is not a widely disseminated capability.

Figure 5 summarizes the number of tools that present each capability. For instance, 13 tools were published from 2001 to 2005, and 7 of them present C1 (Interoperability), 6 of them have C2 (Code generation), and so on. It is observed there was a higher investment in these tools along a decade (from 2006 to 2015), when 76 tools were published, concerning in making them available containing mainly C1 (Interoperability/Integration), C2 (Code generation), C4 (Consistency checking), and C5 (Multiple views). In the last 3 years and a half (2016 to 2019), 13 tools were published with quite similar concerns with regard to capabilities compared to a decade ago. Finally, we can observe that along the 25 years developing variability tools, the amount of attention to each capability has been kept; in general, C1 (Interoperability/Integration), C2 (Code generation), C4 (Consistency checking), and C5 (Multiple views) have been the focus. However, important capabilities to the current software processes (which

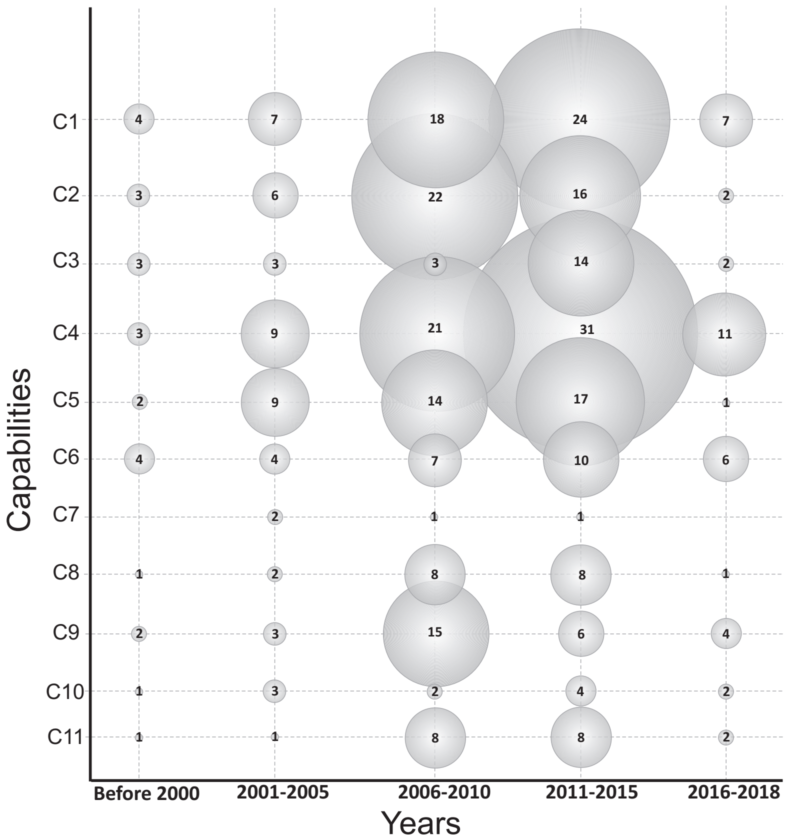

Figure 4: Bubble plot of tools capabilities by period of year

encompass distributed teams developing large, complex systems) have been disregarded, mainly C7 (Collaborative support) and C10 (Scalability). More specifically, while only 3.8% and 11.2% of tools present C7 and C10, respectively, on average 53.5% tools present C1, C2, C4, and C5.

6 Findings

This section discusses whether there is a gap between what the industry requires in variability tools and what in fact exists in the literature. Figure 6 puts together results of our survey (i.e., *percentage of practitioners who indicated capabilities C1 to C11 as missing*) and findings through our SMS (i.e., *percentage of tools that present such capabilities*). Although both quantities refer to different things, this figure is important to show a comparison with regard to each capability, as discussed below:

C1 - Interoperability/Integration: While practitioners claim tools fail to integrate with new technologies (e.g., microservices and cloud computing) and do not support all development life cycle (from requirement engineering to testing and maintenance), only a few tools (more specifically, pure::variants and Gears) in fact provide this large integration covering the entire development processes. Although 56% of tools present some integration with other tools, such integration refers to specific tools like for modeling and consistency analysis. A more complete integration must be further included in variability tools to deal with the entire development life cycle of the current and future large, complex software systems. For instance, The work of [Damiani et al., 2018a, Damiani et al., 2018b] provides interoperability support of software product variants in different development life cycle activities. The OASIS standard on interoperability[12] could contribute to providing interoperability to variability tools, since it encompasses several guidelines to mitigate interoperability and portability in software implementations.

C2 - Code generation and **C9 - Model transformation:** Being two capabilities closely related, they were also most pointed out by practitioners. A good portion of the tools already presents them (46.7% and 29% to C2 and C9, respectively), while 19.2% of tools present both capabilities.

Hence, in more general analysis, there is still room to adequately address these capabilities in an integrated way, including the management of the variability dynamically after deployment. More recently, template-based code generators have been taken into consideration for modeling variability and generating single source code from different abstraction levels/layers [Roth et al., 2016, Syriani et al., 2018]. In addition, feature patterns might be used to generate source code from variability models via transformation [Strüber et al., 2018]. Product model derivation can also be achieved by applying model transformation techniques based on model element relationships, as explored in [Taentzer et al., 2017].

C3 - Impact analysis and **C6 - Traceability:** Most tools (23.4% and 29.0% of the tools for C3 and C6, respectively) do not provide these important capabilities, what is in some extent aligned to the impression of practitioners. Both capabilities working together could possibly reduce software development cost/effort, by better following the impact of any changes in variability along the whole development life cycle (not only among a small number of artifacts as currently occur). Therefore, these are in fact required capabilities that could leverage the completeness of variability tools applied in a large scenario of software development. Traceability techniques could

[12] `https://www.oasis-open.org/policies-guidelines/interoperability-guidelines`

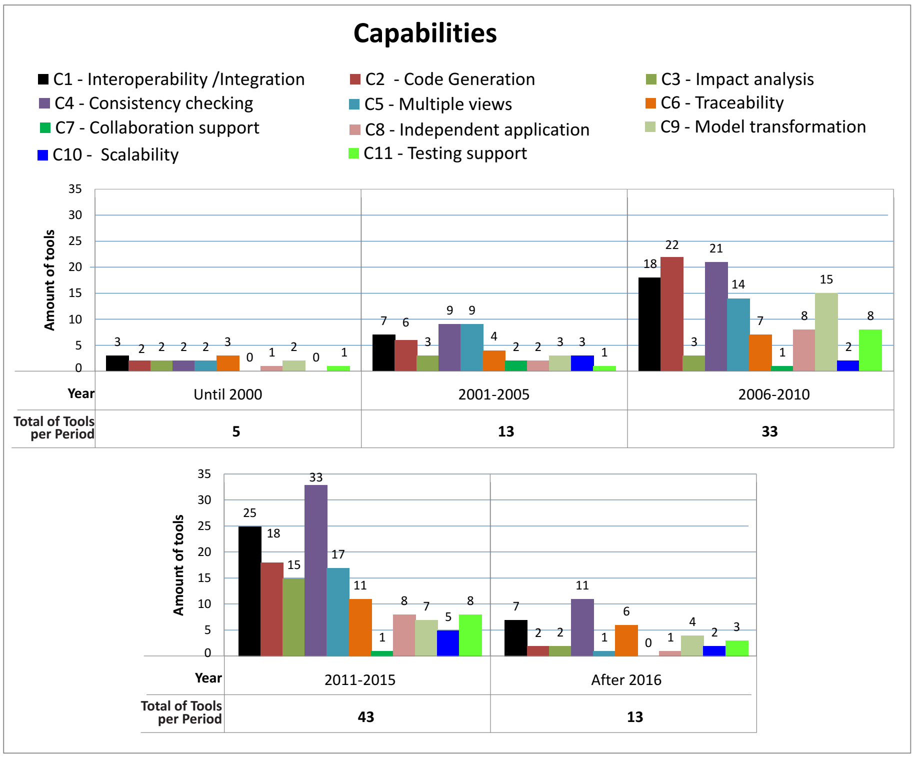

Figure 5: Evolution of software variability tools that exhibit the capabilities C1 to C11

be more systematically explored to link different variability assets, supporting impact analysis, and bridging the semantic distance among such assets in different abstraction levels [Jirapanthong and Zisman, 2009]. In addition, traceability shall also go beyond supporting the evolution of software systems after their deployment. Traceability strategies, such as the creation of links, metamodeling, model transformation, feature tagging, mapping tables, package-merge mechanism, and trace recovery, should be more explored to promote such capability in variability tools [Vale et al., 2017].

C4 - Consistency checking: While practitioners indicated the need for this capability in variability tools, it is the most recurrent one in tools found in our SMS, i.e, in 71%. Trying to understand this discrepancy, we found out five tools mentioned by practitioners (Covamof-VS, DecisionKing, PlugSPL, Visit-FC, and V-Manage) do not have consistency checking and, therefore, these practitioners indicated it as a missing capability. We believe in the relevance of these tools (as they are academic tools known/used in industry) and, hence, these and other tools missing this capability could invest the effort to evolve them, including the constraint management at runtime [Bosch et al., 2015, Berger et al., 2013]. Variability consistency checking rules [Kim and Kim, 2013] and flexible, scalable mechanisms (as in DOPLER tool [Vierhauser et al., 2010]) could be an inspiration to other tools.

C5 - Multiple views: Most tools (40.2%) provide more than one view, but most of them still require multiple views and viewpoints considering different stakeholders involved in the development processes, such as architects, analysts, developers, testers, and so on. Hence, the impression of practitioners regarding this capability should be considered to achieve more effective tools with more powerful representation/visualization techniques in a scenario of the entire software development. Several different visualization techniques are the potential to be explored for different roles and perspectives. For instance, data analytics for the project manager perspective [Smiley et al., 2015], information retrieval [Santos et al., 2012] and scalable visualizations [Cross et al., 1998] both for the software architect perspective are some examples that deserve more attention.

C7 - Collaborative support, **C8 - Independent application**, and **C11 - Integration with a testing tool:** In the current scenario of distributed, global development, support to collaborative work is indispensable in any tools, in particular, when dealing with large, complex systems where there is a tangle of relations among apparently distinguished parts. That is also especially true when considering the capability of independence of variability tools, where such diverse teams can adopt different development tools, platforms, and environments; adoption of dependent variability tools can make their large adoption impossible. In the same perspective, the test of large systems is a current real challenge and whether variability tools can link smoothly to testing tools, taking advances of the power of testing tools to generate test cases to test artifacts (in particular, feature models) and, at the same time, provide information to testing tools to test software products would be in fact a more perfect scenario. Examples such as SPLOT [Pereira et al., 2013], which provides collaborative support for multiple users to configure a single product, should be considered.

C10 - Scalability: Scalability of variability models should be a key concern of the tools in the development of large, distributed, and complex software systems. Solutions to deal with a huge amount of combinations of variants, their relationship, and even constraints should be still deeply investigated and made available in variability tools. In a recent work [Pett et al., 2019], it is proposed the scalability analysis for a large number of features based on product sampling algorithms and constraint-based solvers such as SAT solvers.

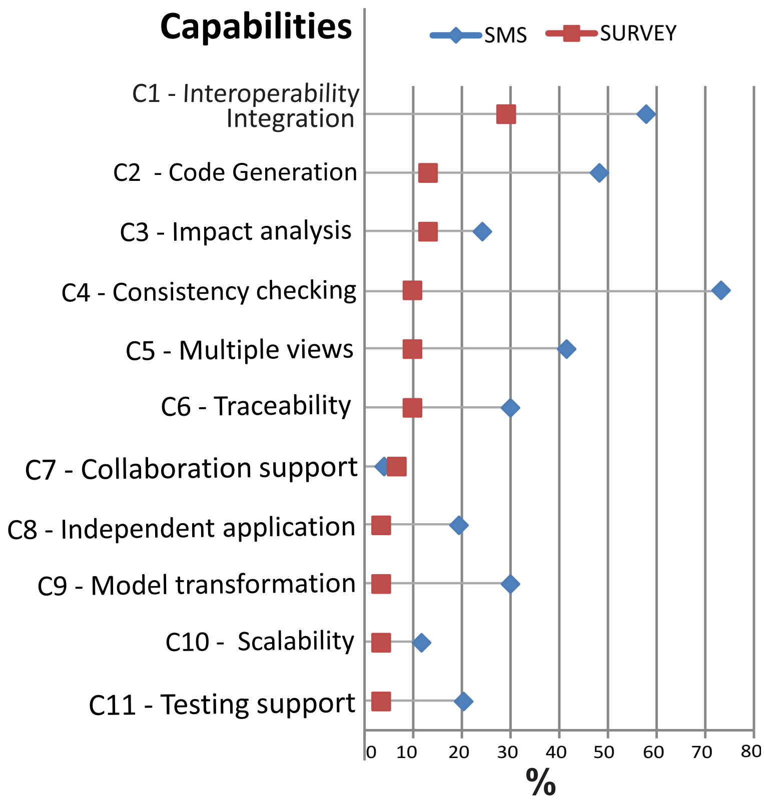

Figure 6: Percentage of missing capabilities in existing software variability tools answered by industry practitioners

Research gaps: Derived from the findings discussed above we summarize here some research gaps that could potentially lead to new research lines to improve the tools presented in this paper.

– **Gap 1:** None of the tools investigated provide support for runtime variability mechanisms, maybe because customers still don't demand these kinds of solutions but quite suitable if we want to start a dynamic software product line or just use a runtime variability manager. Only some recent variability languages like VEL (Variability eXchange Language) suggest runtime binding of variants but not dynamic changes in the structural variability.

- **Gap 2:** Visualization capabilities to show large feature models is still a big challenge. Tools are able to show large feature models as a tree-like structure but many times these are unmanageable when configuring hundreds of features. Therefore, advanced visualization capabilities are still needed.
- **Gap 3:** Interoperability between different tools is still needed as we have different data formats to store feature models and other software artifacts. Hence, we still need a common language to capture any kind of variability construct that can be supported by most used tools.
- **Gap 4:** Reducing the gap between variability modeling and implementations needs to be improved, and not only provide skeletons of features but generate code in a more complete manner. Therefore, we need to reduce the gap between configuration and implementation cycles.
- **Gap 5:** Collaborative capabilities between to support concurrent updates of feature models is a feature not supported by many tools and useful for distributed teams adopting a multi-product line approach.

7 Threats to Validity

Results obtained by this study might have been affected by some threats:

Construct Validity: with respect to our survey, we minimized threats to this validity by designing a focused, multiple-choice questionnaire with a text area to the addition of extra information from respondents. We also thoroughly chose all potential participants, assuring to get relevant answers coming from industrial practitioners or researchers with high expertise in variability tools.

To minimize this threat during our SMS, we systematically followed the guidelines found in [Petersen et al., 2015, Kitchenham and Charters, 2007]. Besides, we used possibly all terms related to "variability management", "SPL", and "tool", which are broadly adopted by the community in studies involving variability tools, to construct our search string. Moreover, we combined both automatic and manual searches to increase the coverage of our search, together with the application of the snowballing technique.

Internal Validity: a threat to internal validity in our survey is that respondents could misunderstand the questions. Hence, before distributing the questionnaire, each question was validated by three experts in variability tools, aiming to mitigate this threat.

The main threats to the internal validity with regard to our SMS refers to the selection of primary studies. To ensure that this SMS is complete and no important study is missing, we used seven publications databases, including all ones recommended for the software engineering area. We also accurately performed a manual search in relevant conference/workshops proceedings and journals. With regard to bias threat, the selection process carried out by one researcher was constantly reviewed by others, together with a rigorous application of the inclusion and exclusion criteria.

External Validity: we did not provide a randomized sample of respondents; however, we mitigated it by selecting outstanding practitioners and researchers from strategic institutions working with variability tools for several years. For the survey, we selected a representative set of questions on variability tools, designing our questionnaire in a way we could preserve the type of information acquired.

With regard to our SMS, we carefully defined a set of inclusion and exclusion criteria to avoid bias in the selection of the primary studies. We also mitigated the potential threat to generalize our results as we came up with a significant sample size of 107 software variability tools. Therefore, we believe our research findings derived from the analysis of this sample size are valuable and complement similar related studies.

Conclusion Validity: it is related to the ability to make correct conclusions. With regard to the survey, our conclusions were based on information provided by respondents in an online questionnaire, which might reflect on beliefs rather than real facts. We mitigated this threat by inviting only experts in variability tools. Another threat is the small number of respondents of our survey; then, aiming at considering their answer valid to a wider population, we only invited those practitioners with deep knowledge on variability tools and years of experience in software development.

To mitigate this threat in our SMS, we applied a rigorous data extraction process based on the guidelines found in [Kitchenham and Charters, 2007, Petersen et al., 2015]. Data from primary studies were extracted using a data extraction form. However, a potential threat might be the reliability of data extraction with respect to types and capabilities of variability tools, as some data extracted had to be interpreted; therefore, to ensure the validity of data extracted and/or interpreted, such data was meticulously reviewed by all authors of this work.

8 Conclusion

With software systems demanding continuous adaptation and faster deployment capabilities, software variability techniques became a valuable technique to support easy configuration and adaptation to changing environments. Much effort has been devoted to advance the state and practice of the tools analyzed in this research and its use by product line processes. However, the increasing complexity of software-intensive systems has brought new challenges to model, manage and visualize large variability models; how this information can be implemented in systems and how to achieve better interoperability between variability models. Therefore, our intention in this research was twofold: (i) support the stakeholders in the selection of adequate variability tools by identifying the important capabilities relevant for industry practitioners; and (ii) identify the existing and missing capabilities in the most popular tools.

Consequently, in this work, we highlighted the perception of industry practitioners about the use of current variability tools in order to identify the important missing capabilities required by industry needs. Thus, the opinions collected during the online survey and direct interviews offer a complementary view to identify some important needs and to improve the most used tools. In particular, the lack of interoperability between tools and a stronger connection between variability modeling and implementation is needed, such as tools like GEARS promote automatic configuration mechanisms for the bill of features in the application engineering cycle. Also, we believe the integration between static and dynamic variability approaches is a characteristic that will be demanded in the next future. In addition, the compliance with software product line and software variability standards is another challenge for companies that want to adopt a product line strategy as they need to know if standardized software processes are supported by current tools. Additionally, we foresee a new generation of software variability tools with enhance visualization capabilities where feature models can be shown not as a whole, but separated into functional areas and easily to be manage.

For the next future, we plan to extend in the future our study comparing the results from more companies with those from SPL Hall of Fame projects, as we were unfortunate to get more results from companies using software variability tools.

Acknowledgments

This research was supported by the Brazilian funding agencies CAPES, Brazil, UEM (code 001), FAPESP (grants: 2016/05919-0, 2017/06195-9, 2018/20882-1, 2019/19730-5) and CNPq (grant: 312634/2018-8), and Spanish research network MCIU-AEI TIN2017-90664-REDT.

References

[Bashroush et al., 2017a] Bashroush, R., Garba, M., Rabiser, R., Groher, I., and Botterweck, G. (2017a). CASE tool support for variability management in software product lines. *ACM Comput. Surv.*, 50(1):14:1–14:45.

[Bashroush et al., 2017b] Bashroush, R., Garba, M., Rabiser, R., Groher, I., and Botterweck, G. (2017b). CASE tool support for variability management in software product lines. *ACM Comput. Surv.*, 50(1):14:1–14:45.

[Benavides et al., 2010] Benavides, D., Segura, S., and Cortés, A. R. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6):615–636.

[Berger et al., 2014] Berger, T., Nair, D., Rublack, R., Atlee, J. M., Czarnecki, K., and Wasowski, A. (2014). Three cases of feature-based variability modeling in industry. In *17th International Conference on Model-Driven Engineering Languages and Systems (MODELS)*, MODELS '14, pages 302–319, Valencia, Spain. Springer International Publishing.

[Berger et al., 2013] Berger, T., Rublack, R., Nair, D., Atlee, J. M., Becker, M., Czarnecki, K., and Wsowski, A. (2013). A survey of variability modeling in industrial practice. In *7th International Workshop on Variability Modelling of Software-intensive Systems (VaMoS)*, VAMOS '13, pages 7:1–7:8, Pisa, Italy. ACM.

[Bhumula, 2013] Bhumula, M. R. (2013). Comparative study and analysis of variability tools. *Computing Resource Repository*, abs/1304.3912:1–35.

[Bosch et al., 2015] Bosch, J., Capilla, R., and Hilliard, R. (2015). Trends in systems and software variability. *IEEE Software*, 32(3):44–51.

[Capilla et al., 2013] Capilla, R., Bosch, J., and Kang, K. C., editors (2013). *Systems and Software Variability Management - Concepts, Tools and Experiences*. Springer.

[Chituc, 2017] Chituc, C. (2017). Interoperability standards for seamless communication: An analysis of domain-specific initiatives. In *OTM*, OTM '17, pages 36–46. Springer International Publishing.

[Cross et al., 1998] Cross, J. H., Hendrix, T. D., Barowski, L. A., and Mathias, K. S. (1998). Scalable visualizations to support reverse engineering: a framework for evaluation. In *Proceedings Fifth Working Conference on Reverse Engineering (Cat. No.98TB100261)*, WCRE '98, pages 201–209.

[Damiani et al., 2018a] Damiani, F., Hähnle, R., Kamburjan, E., and Lienhardt, M. (2018a). Interoperability of software product line variants. In *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, pages 264–268, New York, NY, USA. ACM.

[Damiani et al., 2018b] Damiani, F., Hähnle, R., Kamburjan, E., and Lienhardt, M. (2018b). *Same Same But Different: Interoperability of Software Product Line Variants*, pages 99–117. Springer International Publishing, Cham.

[Hilliard, 2010] Hilliard, R. (2010). On representing variation. In *4th European Conference on Software Architecture (ECSA)*, ECSA '10, pages 312–315, Copenhagen, Denmark. ACM.

[Ihme et al., 2014] Ihme, T., Pikkarainen, M., Teppola, S., Kääriäinen, J., and Biot, O. (2014). Challenges and industry practices for managing software variability in small and medium sized enterprises. *Empir. Softw. Eng.*, 19(4):1144–1168.

[ISO/IEC, 2012] ISO/IEC (2012). Software and systems engineering - tools and methods for product line requirements engineering (ISO/IEC 26551).

[ISO/IEC, 2013a] ISO/IEC (2013a). Software and systems engineering - Reference model for product line engineering and management (ISO/IEC 26550).

[ISO/IEC, 2013b] ISO/IEC (2013b). Software and systems engineering - Tools and methods for product line technical management (ISO/IEC 26555).

[Jirapanthong and Zisman, 2009] Jirapanthong, W. and Zisman, A. (2009). Xtraque: traceability for product line systems. *Software & Systems Modeling*, 8(1):117–144.

[Kim and Kim, 2013] Kim, J. A. and Kim, S. (2013). Consistency checking rules of variability in software product lines. In *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, BWCCA '13, pages 595–597.

[Kitchenham and Charters, 2007] Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Keele University and Durham University Joint Report.

[Lisboa et al., 2010] Lisboa, L. B., Garcia, V. C., Lucrédio, D., de Almeida, E. S., de Lemos Meira, S. R., and de Mattos Fortes, R. P. (2010). A systematic review of domain analysis tools. *Information and Software Technology*, 52(1):1–13.

[Mahdavi-Hezavehi et al., 2013] Mahdavi-Hezavehi, S., Galster, M., and Avgeriou, P. (2013). Variability in quality attributes of service-based software systems: A systematic literature review. *Inf. Softw. Technol.*, 55(2):320–343.

[Molléri et al., 2016] Molléri, J. S., Petersen, K., and Mendes, E. (2016). Survey guidelines in software engineering: An annotated review. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '16, pages 58:1–58:6, New York, NY, USA. ACM.

[Pereira et al., 2015] Pereira, J. A., Constantino, K., and Figueiredo, E. (2015). A systematic literature review of software product line management tools. In *14th International Conference on Software Reuse for Dynamic Systems in the Cloud and Beyond (ICSR)*, ICSR '15, pages 73–89, Miami, FL, USA. Springer International Publishing.

[Pereira et al., 2013] Pereira, J. A., Souza, C., Figueiredo, E., Abilio, R., Vale, G., and Costa, H. A. X. (2013). Software variability management: An exploratory study with two feature modeling tools. In *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse*, SBCARS '13, pages 20–29.

[Petersen et al., 2015] Petersen, K., Vakkalanka, S., and Kuzniarz, L. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information & Software Technology*, 64(C):1–18.

[Pett et al., 2019] Pett, T., Thüm, T., Runge, T., Krieter, S., Lochau, M., and Schaefer, I. (2019). Product sampling for product lines: The scalability challenge. In *Proceedings of the 23rd International Systems and Software Product Line Conference - Volume A*, SPLC '19, pages 78–83, New York, NY, USA. ACM.

[Pohl et al., 2005] Pohl, K., Böckle, G., and van der Linden, F. J. (2005). *Software Product Line Engineering: Foundations, Principles and Techniques*, volume 26. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

[Pohl et al., 2013] Pohl, R., Stricker, V., and Pohl, K. (2013). Measuring the structural complexity of feature models. In *28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, ASE '13, Silicon Valley, CA, USA.

[Raatikainen et al., 2019] Raatikainen, M., Tiihonen, J., and Männistö, T. (2019). Software product lines and variability modeling: A tertiary study. *J. Syst. Softw.*, 149:485–510.

[Roth et al., 2016] Roth, A., Greifenberg, T., Müller, K., Rumpe, B., Schulze, C., and Wortmann, A. (2016). Modeling variability in template-based code generators for product line engineering. In *Lecture Notes in Informatics*, Modellierung '16, pages 141–156.

[Santos et al., 2012] Santos, W. B., de Almeida, E. S., and de L. Meira, S. R. (2012). Tirt: A traceability information retrieval tool for software product lines projects. In *2012 38th Euromicro Conference on Software Engineering and Advanced Applications*, SEAA '12, pages 93–100.

[Sierszecki et al., 2014] Sierszecki, K., Steffens, M., Hojrup, H. H., Savolainen, J., and Beuche, D. (2014). Extending variability management to the next level. In *18th Software Product Line Conference (SPLC)*, SPLC '14, pages 320–329, Florence, Italy. ACM.

[Smiley et al., 2015] Smiley, K., Schmidt, W., and Dagnino, A. (2015). Evolving an industrial analytics product line architecture. In *Proceedings of the 19th International Conference on Software Product Line*, SPLC '15, pages 263–272, New York, NY, USA. ACM.

[Strüber et al., 2018] Strüber, D., Peldzsus, S., and Jürjens, J. (2018). Taming multivariability of software product line transformations. In Russo, A. and Schürr, A., editors, *Fundamental Approaches to Software Engineering*, FASE '18, pages 337–355, Cham. Springer International Publishing.

[Syriani et al., 2018] Syriani, E., Luhunu, L., and Sahraoui, H. (2018). Systematic mapping study of template-based code generation. *Computer Languages, Systems & Structures*, 52:43 – 62.

[Taentzer et al., 2017] Taentzer, G., Salay, R., Strüber, D., and Chechik, M. (2017). Transformations of software product lines: A generalizing framework based on category theory. In *2017 ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, MODELS '17, pages 101–111.

[Vale et al., 2017] Vale, T., de Almeida, E. S., Alves, V., Kulesza, U., Niu, N., and de Lima, R. (2017). Software product lines traceability. *Inf. Softw. Technol.*, 84(C):1–18.

[Vierhauser et al., 2010] Vierhauser, M., Grünbacher, P., Egyed, A., Rabiser, R., and Heider, W. (2010). Flexible and scalable consistency checking on product line variability models. In *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ASE '10, pages 63–72, New York, NY, USA. ACM.

[Wohlin, 2014] Wohlin, C. (2014). Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *8th International Conference on Evaluation and Assessment in Software Engineering (EASE)*, EASE '14, pages 1–10, London, England. ACM.

[Wohlin et al., 2012] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A. (2012). *Experimentation in Software Engineering: An Introduction*. Springer-Verlag, Norwell, MA, USA, 2 edition.