1. State of the art

As previously stated, software components are some of the most important SPL artifacts, as they are the base on which each new application of the product line is produced. SPL components are used throughout several SPLE stages such as binding, development, testing, evolution, and derivation, among others. Due to the way SPL components are integrated into several of those stages, we decided to develop a systematic mapping study (SMS) on SPL implementation. This SMS provides an overview of the processes, methods, and tools used to carry out SPL implementation, but also provides details on the role of the SPL components in the entire process.

An SMS is a form of a secondary study aiming to provide a comprehensive overview of a certain research topic, to identify gaps in the research, and to collect evidence in order to guide researchers and practitioners in their current or future work (Wohlin *et al.*, 2013). It allows all available studies in a domain to be analyzed at a high level thereby answering broad research questions regarding the current state of the research on a topic. Another form of secondary study is a systematic literature review (SLR), which aims to identify, evaluate, and interpret all available studies to answer specific research questions (usually in the form: Is technology/method A better than B or not?), and requires more in-depth analysis (Kitchenham & Charters, 2007). We decided to conduct an SMS instead of an SLR because we wanted to provide a complete understanding of the current studies developed in the SPL implementation field, covering its different stages and answering high-level questions about SPL implementation.

The main goals of this SMS are:

- (G1) to provide basic publication information through a demographic method and to assist researchers in identifying the most appropriate sources of research information on SPL implementation.

- (G2) to assist researchers and practitioners in identifying the range of methods and mechanisms currently available for an SPL implementation (including SPL component implementation).
- (G3) to identify principal research trends in the literature and highlight active research topics that require more future work.

To achieve these objectives, this SMS will answer a selection of the research questions presented in Section 1.2.1.

The remainder of this section is organized as follows: Section 1.1 presents the related work; Section 1.2 describes the systematic mapping process, the definition of the research goals and questions, the search strategy, and the search conduction are presented; Section 1.3 presents the data extraction; Section 1.4 presents the answers to questions about research goal G1 – Publication; Section 1.5 presents the answers to questions about research goal G2 – SPL implementation; Section 1.6 presents the answers to questions about research goal G3 – Topics and trends; Section 1.7 discusses the threats to validity; and finally, Section 1.8 concludes this study and recommends the direction of future work.

Note: It is important to highlight that the references used in the SMS can be found in Appendix A. Therefore, some of these references will be presented in this section with the IEEE format, and they will be preceded by the letter S.

1.1 Related work

During the last decade, many authors have carried out SMS, SLR and surveys on SPL [S11, S12, S13, S33, S54, S55, S71, S77, S82, S87], with these contributions covering many aspects of the field of SPL. However, most of them involve a specific part or even a specific method for the SPL implementation process. As a result, a complete understanding of SPL implementation is not given. Therefore, the research questions posed in this SMS are legitimate and previous studies have not yet answered them. A summary of the currently available SMS, SRL, and survey studies is presented below.

Lee *et al.* [S11] presented a survey framework that consists of eight SPL specific testing perspectives and compared and analyzed the contributions of selected studies. Neto *et al.* [S12] developed a systematic mapping study on Testing in SPL in which 120 studies were evaluated. They found a huge amount of approaches that handle different and specific

aspects in the SPL testing process, however, the quantity of approaches makes comparing studies a difficult task. Through this study, they were able to identify which activities are handled by the existing approaches as well as understanding how researchers are developing work on SPL testing.

Laguna and Crespo [S13] carried out a study which aimed to survey the existing research on the reengineering of legacy systems like SPLs and the refactoring of existing SPLs. Guided by several parameters, 74 papers were selected and classified. The results of the study indicate that the initial works focused on the adaptation of generic reengineering processes to SPL extraction. Several trends were detected in the research: the integrated or guided reengineering of (typically object-oriented) legacy code and requirements; specific aspect-oriented or feature-oriented refactoring of SPLs, and refactoring for the evolution of existing product lines. Most papers included academic or industrial case studies, although only a few were based on quantitative data. Montalvillo and Díaz [S71] conducted a mapping study on SPL evolution that included 107 articles. They developed a classification schema that included four facets: evolution activity, product-derivation approach, research type, and asset type. The results show that regarding the evolution activity, “Implement change” (43%) and “Analyze and plan change” (37%) were the most covered contributions.

Mohabbati *et al.* [S33] developed a systematic mapping study on the combination of service-orientation and SPLE. In this SMS, 81 primary studies were selected. Their research focused on service variability modeling, service identification, service reuse, service configuration and customization, dynamic software product lines, and adaptive systems. The results show that SPLE approaches, especially feature-oriented approaches for variability modeling, have been applied to the design and development of service-oriented systems. Service-orientation is employed in software product line contexts for the realization of product lines to reconcile the flexibility, scalability, and dynamism in product derivations thereby creating dynamic software product lines.

Afzal *et al.* [S54] conducted a literature review of both research and industrial artificial intelligence applications for SPL configuration issues. They found 19 relevant research papers which employ traditional artificial intelligence techniques on small feature sets with no real-life testing or application in industry. Finally, they showed that only 2 standard industrial SPL tools employ artificial intelligence in a limited way to resolve inconsistencies.

Méndez-Acuna *et al.* [S55] developed a literature review in which they reported an attempt to organize the literature on language product line engineering. More precisely, they proposed a definition for the life-cycle of language product lines, and they used it to analyze the capabilities of current approaches (38 studies were included). In addition, they mapped each approach and the technological space supported by it.

Vale *et al.* [S77] conducted a systematic mapping study to investigate the state-of-art of the SPL traceability area, in which 62 primary studies were identified. The results showed that the common strategies for systematizing traceability were metamodeling, different representation structures, model transformations, formal methods, and trace recovery techniques. Most strategies focus on the trace creation activities, with a lack of planning, maintenance, and use of SPL traces.

Dos Santos Rocha and Fantinato [S82] performed an SLR with four research questions formulated to evaluate PL approaches for BPM (63 papers were selected). The results showed that the PL approaches found for BPM only partially cover the BPM lifecycle, not taking into account the last phase which restarts the lifecycle. Therefore, the results indicate that PL approaches for BPM are still at an early stage and are gaining maturity.

Mazo *et al.* [S87] carried out a literature review. Their objective was to identify and analyze the different ways for improving ERP engineering issues with the methods, techniques, and tools provided by PLE. Their literature review analyzed six research papers and found that there is still a lack of interest in addressing ERP engineering issues with the product line strategy.

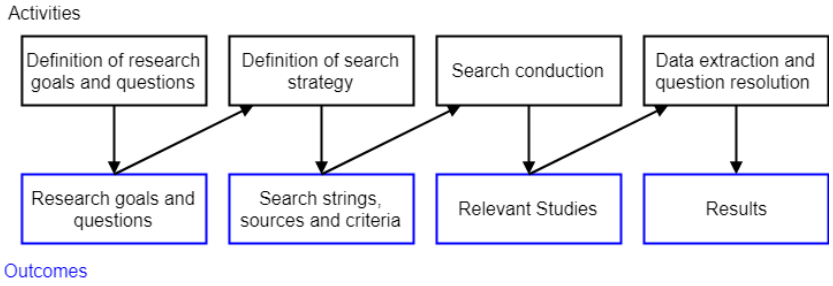
Pereira *et al.*, (2015) developed an SLR which analyzed the available literature on SPL management tools. In this study, 52 papers were included. They identified 41 tools in the literature that provide support to at least one SPL management phase.

Finally, Marimuthu & Chandrasekaran (2017) conducted a systematic mapping study of existing systematic studies of software product lines (tertiary study). They analyzed 60 relevant studies to highlight the SPL research topics, type of published reviews, active researchers and publication forums.

1.2 Systematic mapping process

As previously mentioned, the method used in this research is an SMS. We applied the mapping studies guidelines proposed by Petersen *et al.* (2008), which compares the methods used in mapping studies and SLR. The specific systematic mapping process reported in this paper was performed based on those guidelines and represented in Figure 1-1 as a sequence of activities and their corresponding outcomes. Even if it is not possible to conduct a mapping study or a literature review in a fully objective manner, the guidelines used in our systematic mapping process on SPL implementation renders the study less subjective thanks to pre-defined data types and criteria that narrow the scope for personal interpretation.

Figure 1-1: The systematic mapping process



The main activities of this systematic mapping process are (i) definition of research goals and questions, (ii) definition of search strategy, (iii) search conduction, and (iv) data extraction and question resolution. The first three activities are described in the next three subsections Section 1.2.1, Section 1.2.2, and Section 1.2.3; the last activity is described in Section 1.3.

1.2.1 Definition of research goals and questions

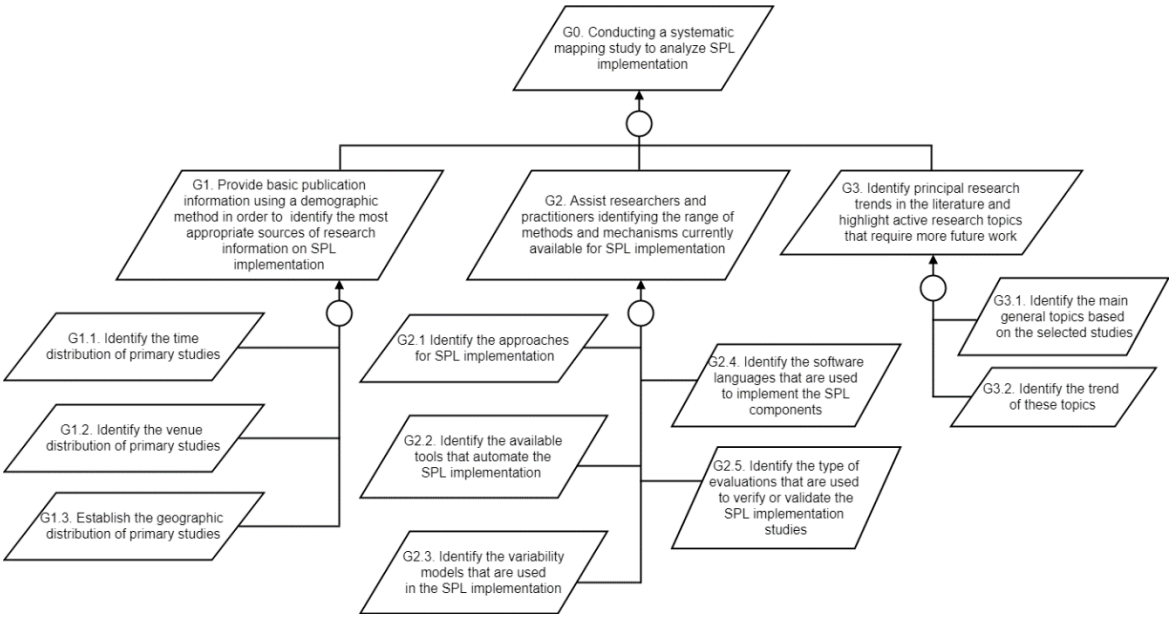
Figure 1-2 shows a KAOS diagram (Heaven & Finkelstein, 2004) that details the research goals of this study. The main research goal (G0) refers to the carrying out of the SMS on SPL implementation. G0 is broken down into three primary sub-goals and these primary sub-goals into secondary sub-goals; the secondary sub-goals will serve as a base for the definition of the research questions.

- Primary sub-goals G1 and G3 represent the researchers' perspective. Their secondary sub-goals provide an overview of how different researchers are dealing

with SPL implementation in different countries, in different laboratories, and where these contributions are available and what the current trends are.

- Primary sub-goal G2 represents the practitioners, industry and developers' perspectives. Its secondary sub-goals provide an overview of current methodologies for implementing an SPL, the available tools, software languages, and evaluations, among others.

Figure 1-2: Systematic mapping study research goals



Based on the previous diagram, we designed 10 answerable and interpretable research questions (SRQ). These research questions will be called SRQ to avoid confusion with the thesis research questions (RQ). Each SRQ is related to a specific secondary sub-goal which is simultaneously related to a primary sub-goal. Table 1-1 shows the 12 research questions and their division.

Table 1-1: Systematic mapping study research questions

Goal	Sub-goal	Research Question
G1	G1.1	**SRQ1:** What is the time distribution of primary studies? **Rationale:** answering this SRQ will help us to understand if this is a trending field and how it has evolved over the years.
	G1.2	**SRQ2:** What is the venue distribution of primary studies?

		Rationale: it is important to know what the authors preferred venues are in order to know where to find relevant papers about this field.
	G1.3	**SRQ3:** What is the geographic distribution of primary studies? **Rationale:** this SRQ will help us to understand what the lead countries and authors in this field are.
G2	G2.1	**SRQ4:** What approaches for SPL implementation are used? **Rationale:** commonly, the base of SPL implementation is a programming paradigm or mechanism which defines the design, implementation, and assembly of the reusable domain components. By answering this SRQ, we can get an overview of how researchers deal with SPL component implementation.
	G2.2	**SRQ5:** What are the available tools that automate SPL implementation? **Rationale:** SPL implementation is a difficult process that involves a lot of activities. The answer to this question will let us know what the available tools and programs are, what processes they automate, and how these tools are used. Based on this, researchers and companies can learn which tools can be used to satisfy their needs.
	G2.3	**SRQ6:** What variability models are most used in an SPL implementation? **Rationale:** This question specifies that system variability is a common field in SPL implementation. Answering this question will let us know what the most used variability models are.
	G2.4	**SRQ7:** What software languages are most used to implement the SPL components? **Rationale:** Commonly companies and developers have preferred software languages that they have used before to develop software products. Knowing the programming languages that are most used by authors and developers to implement SPL components will serve to discover what the matureness of the technologies and mechanisms using those languages are.
	G2.5	**SRQ8:** What type of evaluations are most used in studies on SPL implementation? **Rationale:** Knowing how the authors evaluate their proposals is useful for future studies.
G3	G3.1	**SRQ9:** What are the main topics of the selected studies? **Rationale:** by answering this SRQ, we can get an overview of what the main topics are in SPL implementation. This information is an important starting point for deepening researchers' topics of interest.
	G3.2	**SRQ10:** What trends have these topics followed over the last years? **Rationale:** trending topics are very valuable; they can lead researchers to focus on future research and new developments. Also, this answer helps to understand what topics authors have been developing recently and what challenges they have had.

1.2.2 Definition of search strategy

Keeping in mind the previous research questions, we defined a set of terms. These terms also consider three subjects: (i) the application domain, (ii) the SPL implementation stage, and (iii) the research perspective.

1. **Application domain:** contextualizes the search; in this case, the search only encompasses documents related to software product lines.
2. **SPL implementation stage:** as was mentioned before, the intention is to obtain a complete overview of SPL implementation. This process is comprised of different stages, and so relevant articles need to be found for each stage to obtain the complete overview. For this SMS, we selected 8 principal stages which range from specification to evolution (specification, modeling, binding, personalization, configuration, assembling, validation, and evolution).
3. **Research perspective:** the present SMS perspective is to obtain information about SPL implementation. Then, three terms were added that will help to refine and to obtain results related to this field.

These elements are consolidated in Table 1-2, and Table 1-3 lists the final derived search strings used to conduct the search. The strings are the result of the combination of three parts: (i) all terms in Table 1-2 – Group 1 (separated by “OR”); (ii) specific terms related to each SPL implementation stage, terms of Table 1-2 – Group 2 (separated by “OR”); and (iii) all terms in Table 1-2 – Group 3 (separated by “OR”).

Table 1-2: Group of terms

Subject	**Term**	**Group**
Application domain	Product line, product family, SPL	1
SPL implementation stage	Specification: (domain engineering, domain requirements, requirements engineering) – Modeling: (variability language, domain design, variability model) – Binding: (feature binding, variability binding) – Personalization: (product personalization, market personalization, software personalization, component personalization) – Assembling: (product assembling, software assembling, component assembling) – Configuration: (product configuration, software configuration, component configuration, application realization, application implementation) – Validation: (product validation, quality assurance, software validation, product	2

	testing, component validation) – Evolution: (product evolution, software evolution, company evolution, component evolution)	
Research perspective	Component implementation, software implementation, product implementation	3

Table 1-3: Resulting search strings

Search strings	**SPL Implementation Stage**	**No.**
(“product line” OR “product family” OR “SPL”) AND (“domain engineering” OR “domain requirements” OR “requirements engineering”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Specification	DS1
(“product line” OR “product family” OR “SPL”) AND (“variability language” OR “domain design” OR “variability model”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Modeling	DS2
(“product line” OR “product family” OR “SPL”) AND (“feature binding” OR “variability binding”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Binding	DS3
(“product line” OR “product family” OR “SPL”) AND (“product personalization” OR “market personalization” OR “software personalization” OR “component personalization”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Personalization	DS4
(“product line” OR “product family” OR “SPL”) AND (“product assembling” OR “software assembling” OR “component assembling”) AND (“software implementation” OR “product implementation” OR “component implementation”)	Assembling	DS5
(“product line” OR “product family” OR “SPL”) AND (“product configuration” OR “software configuration” OR “component configuration” OR “application realization” OR “application implementation”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Configuration	DS6
(“product line” OR “product family” OR “SPL”) AND (“product validation” OR “quality assurance” OR “software validation” OR “product testing” OR “component validation”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Validation	DS7
(“product line” OR “product family” OR “SPL”) AND (“product evolution” OR “software evolution” OR “company evolution” OR “component evolution”) AND (“component implementation” OR “software implementation” OR “product implementation”)	Evolution	DS8

In addition to the search strings, we established the search sources used to find the primary studies which are shown in Table 1-4. According to Dyba *et al.* (2007), these databases are efficient for conducting systematic studies in the context of software engineering. Furthermore, these databases have also been considered in another SMS (Laguna & Crespo, 2013). After a first consolidation of the results, other databases were considered as part of a second phase (Google Scholar and Citeseerx) to try to find additional results that could offer useful material.

For each database, we applied a "trial search". This trial search consisted of introducing the first derived search string (DS1) into each database search form, and we checked if the results were as expected. If less than 10 documents were returned or if there were millions of results with inconsistent articles, then, those databases were discarded (DB6, DB7, and DB8 were discarded). Finally, we introduced each resulting search string into each selected search source and collected the results.

Table 1-4: Selected search sources

#	URL	Source	Selected
DB1	http://dl.acm.org/	ACM DL	Yes
DB2	http://ieeexplore.ieee.org/	IEEE Explore	Yes
DB3	http://www.sciencedirect.com/	ScienceDirect	Yes
DB4	http://www.springer.com/la/	Springer	Yes
DB5	https://scholar.google.com	Google Scholar	Yes
DB6	http://www.scopus.com	Scopus	No (after trial search)
DB7	http://citeseerx.ist.psu.edu	Citeseerx	No (after trial search)
DB8	http://www.isiknowledge.com	Web of Science	No (after trial search)

Inclusion/exclusion criteria

Before conducting the search, the following restrictions and quality criteria for including/excluding publications were defined. These criteria were developed with the intention of finding the most relevant papers to solve the research questions and to exclude papers which do not fit this field and do not allow the research questions to be solved.

- **Restriction R1:** The study only includes papers available in electronic form. Books were analyzed based on information available online and using the hard copy versions.
- **Restriction R2:** Only publications written in English were included.
- **Restriction R3:** Articles related to the topic of this paper published between 1st January 2000 and 31st March 2017 were included.

- **Quality criterion Q1:** Each publication was checked for completeness. Publications containing several unsupported claims or that frequently referred to existing work without providing citations were excluded.
- **Quality criterion Q2:** Works by the same authors with very similar content were included and grouped under the same category (method).

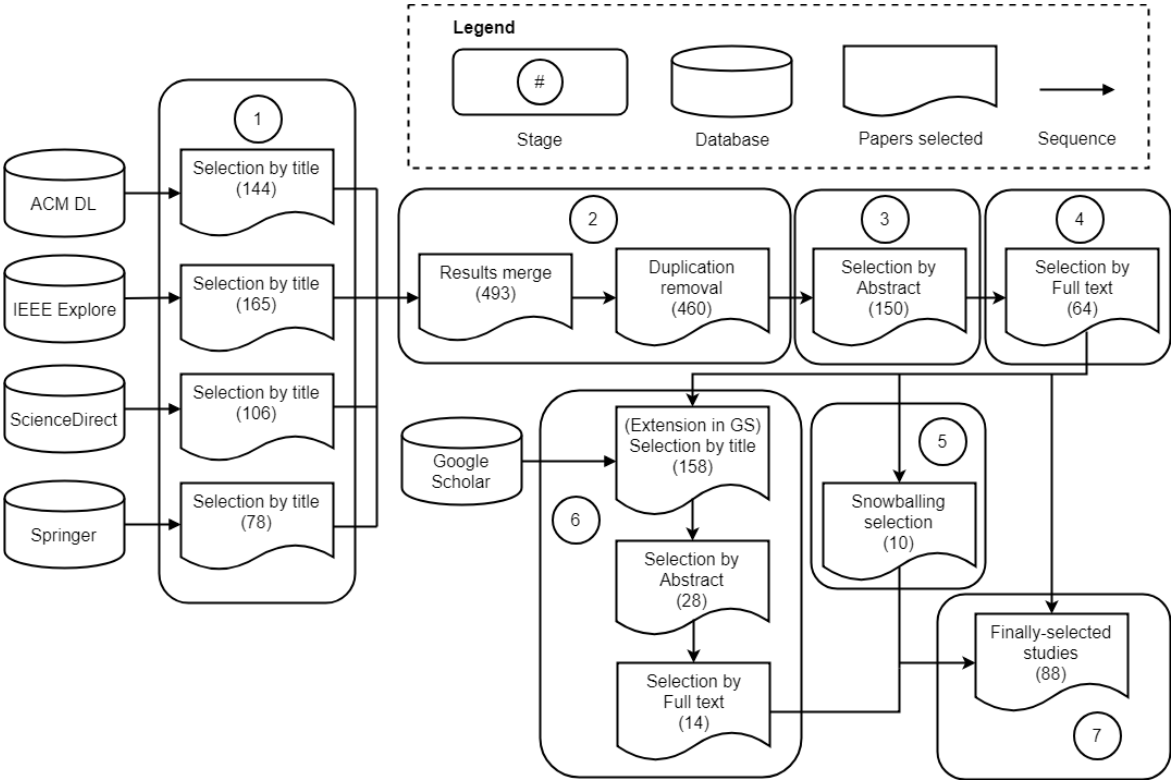
1.2.3 Definition of search conduction

The search conduction was composed of seven stages based on the study by Li *et al.* (2015) work: (i) selection by title, (ii) first results merge, (iii) selection by abstract, (iv) selection by full text, (v) snowballing, (vi) search extension in Google Scholar, and (vii) final results merge. These stages (see Figure 1-3) used the previous search strings and criteria, and they are presented below:

1. **Selection by title:** the search conduction started by using the search strings in four search sources (ACM DL, IEEE Explore, ScienceDirect and Springer), then candidate studies were selected based on the title. Restriction R1, R2, and R3 were applied in this step.
2. **First results merge:** all candidate studies were merged (493 at this point) and duplicated studies were removed (33 studies were removed).
3. **Selection by abstract:** the next stage analyzed the candidate studies' abstracts; at this point, 150 candidate studies were selected.
4. **Selection by full text:** the previous studies' full texts were analyzed and as a result 64 studies were selected. Quality criteria Q1 and Q2 were applied in this step.
5. **Snowballing:** in order not to miss any potentially relevant studies, we applied the "snowballing" technique to find more connected studies by checking the references of the selected studies. This process could be iterative as snowballing and could be repeated in the newfound studies. However, only the first iteration was applied, and 10 new studies were found.
6. **Search extension in Google Scholar:** parallel to stage 5, we extended the search by looking in Google Scholar, we used the search strings and made a first scan by title (Restriction R1, R2, and R3 were applied in this step), then, a second scan by abstract, and finally a third scan by full text (Quality criteria Q1 and Q2 were applied in this step). 14 new studies were found.

7. **Final results merge:** at the end, we merged the selected studies from stages 4 and 5 and 6, and 88 relevant studies were selected. These relevant studies are listed in Appendix A.

Figure 1-3: Study selection stages



1.3 Data extraction

The data extraction process consisted of collecting key information about relevant studies that will be the base to answer the research questions. We created a spreadsheet and for each study we recorded 18 pieces of data (see Table 1-5), the description of each data point is also included in the next table (this description explains in detail the kind of data to be collected). Lastly, relevant research questions were assigned to each data point; this is because some of this data serves to answer one or multiple research questions.

Table 1-5: Studies data extraction

#	Data item name	Description	Relevant SRQ
D1	Article name	The name of the article	None
D2	Article year	The year in which the article was published	SRQ1

D3	Type of publication	The publication type, such as journal, conference, workshop, symposium, or book chapter	SRQ2
D4	Publication name	The name of the journal, conference, workshop, symposium or book chapter where the article was published	SRQ2
D5	Publication venue	The name of the publication venue of the study	SRQ2
D6	Authors	The name of all authors that participate in the study	SRQ3
D7	Country	The main author's affiliation country	SRQ3
D9	Implementation approach	The programming paradigm or approach used to design, implement, and assemble the SPL components	SRQ4
D10	Implementation stage	The SPL implementation stage that was covered in the article	SRQ5
D11	Tools	The name of the tools presented or used in the article	SRQ5
D12	IDE	The integrated developed environments presented or used in the article	SRQ5
D13	Variability models	The variability models presented or used in the article	SRQ6
D14	Provide example with programming code	Yes or no depending on if the article provides an example with programming code	SRQ7
D15	Software languages	The software languages presented or used in the article to implement the SPL components	SRQ7
D16	Evaluation type	The type of evaluation presented or used in the article	SRQ8
D17	Example type	The type of example presented in the article	SRQ8
D18	Topics and Trends	A general topic discussed in the article and/or trends	SRQ9-SRQ10

1.4 Resolving questions about research goal G1 – Publication

The first analysis after the search conduction gives us a global overview of the time distribution and the diversity of the sources in this research field (which provides information to answer SRQ1 and SRQ2). Implementation of SPL has been a relevant topic over the last decade (see Figure 1-4); at the beginning of the twenty-first century there were a few studies in this field, and in 2016 publication peaked with 10 studies published. 2017 shows only 4 publications, but the time-frame between the papers are sent to be published and their final publication should be considered, as the search conduction was carried out between March and April of 2017.

Figure 1-4: Temporal distribution of the sources

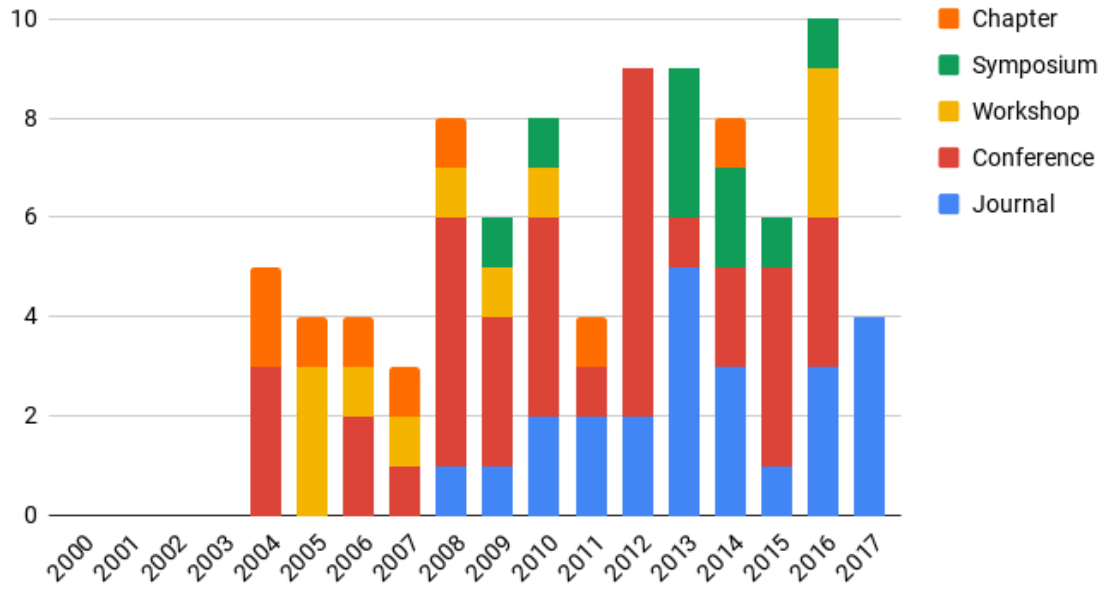
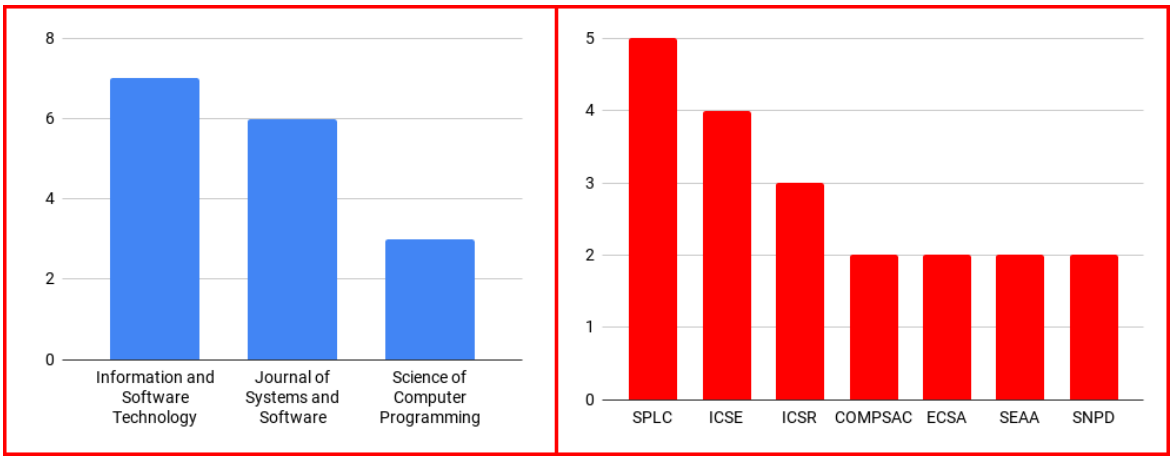


Figure 1-4 shows some interesting information. Most of the publications were presented at conferences or similar (symposiums and workshops) with a total of 56 publications (64%). The second preferred medium is journals with 24 publications (27%) and finally book chapters with 8 publications (9%).

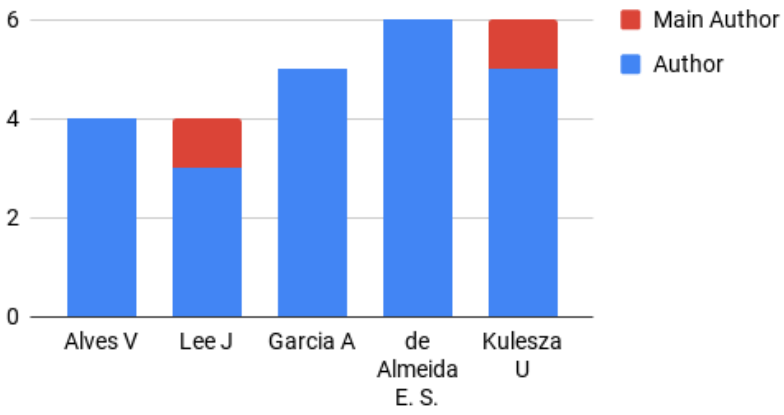
Another important result is related to conferences and journals that have published the most studies (see Figure 1-5). For journals, *Information and Software Technology* with 7 publications was the most used; second, *Journal of Systems and Software* with 6 and third, *Science of Computer Programming* with 3. For conferences, International Software Product Line Conference (SPLC) with 5 publications was the most used; second, International Conference on Software Engineering (ICSE) with 4 and third, International Conference on Software Reuse (ICSR) with 3 publications.

Figure 1-5: Data on most frequent journals and conferences



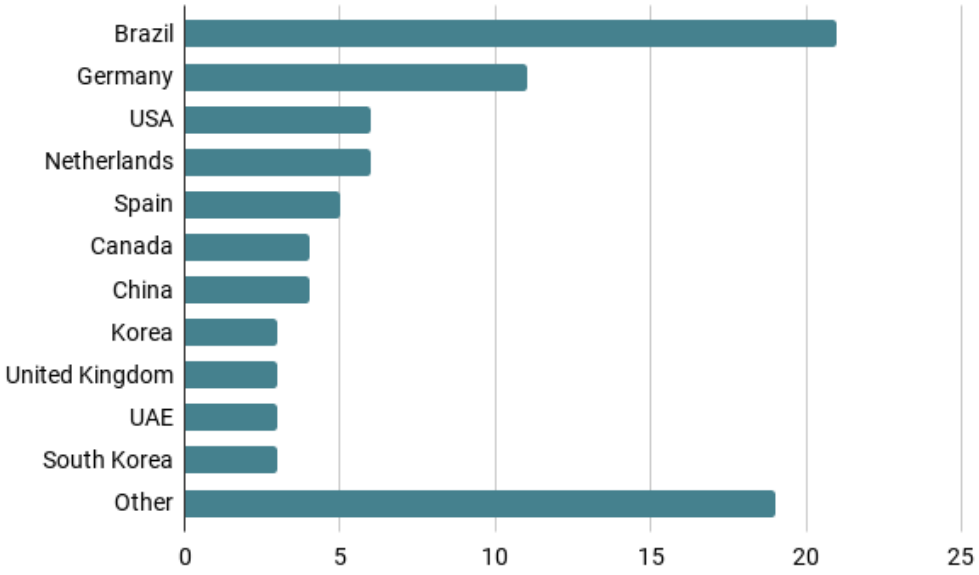
Top authors in the research field are presented in Figure 1-6, showing the number of contributions from each author as a main author or secondary author. Uirá Kulesza (Kuleza U) from Universidad Federal de Río Grande del Norte (Brazil) and Eduardo Santana de Almeida (de Almeida E.S.) from Universidade Federal da Bahia (Brazil) are the top contributors with 6 publications each, followed by Alessandro Garcia (Garcia A) from Pontificia Universidad Católica de Río de Janeiro (Brazil) with 5 publications. Finally, Jaejoon Lee (Lee J) from Lancaster University (UK) and Vander Alves (Alves V) from Universidade de Brasília (Brazil) present 4 publications each (this analysis helps to answer the SRQ3).

Figure 1-6: Data on most frequent authors'



Based on the previous figure, Brazil leads as the country with the most top contributors. This is consistent with the results presented in Figure 1-7. This figure provides a list of the top countries that each publication's main authors are affiliated to (SRQ4 research question). It shows Brazil also leads with 21 publications (24%), followed by Germany with 11 publications (13%), and USA and Netherlands with 6 publications each.

Figure 1-7: Data on each publication's main authors' affiliations



1.5 Resolving questions about research goal G2 – SPL implementation

After the search conduction was carried out, the second analysis focused on the main topic of the SMS (SPL implementation), which provides information for answering SRQ4 to SRQ8. First, we summarize the results of the SPL implementation approaches; second, we present the results in order to answer the research questions.

1.5.1 Summary of SPL implementation approaches

We divided the studies based on their SPL implementation approach (See Table 1-5 – D9). We grouped these studies as follows: (i) aspect-oriented programming (AOP), (ii) service-oriented architecture (SOA), (iii) annotative approaches, (iv) feature-oriented programming (FOP) and delta-oriented programming (DOP), (v) other approaches, (vi) mixed

approaches, (vii) not specified. After summarizing each approach, we will show an example of component implementation with some of the approaches listed above. These approaches also describe the way in which the SPL components must be implemented.

Below, the results of the information collected for each of the previous groups are presented. To organize the information, we created a table structure that summarizes the collected information (based on Table 1-5). This structure records:

- **SPLE stage** refers to the software product line engineering stage, that was covered in the study such as modeling, binding, implementation, and evolution. Sometimes a study can cover multiple SPLE stages, in these cases, we recorded the information as “Multiple”.
- **Evaluation** refers to the type of evaluation applied in the study such as running examples, experiments, and comparisons.
- **Specific example** refers to the type of example presented in the study.
- **Tool support** refers to the tools, libraries or software used or mentioned in the study.
- **Software language** refers to the software languages used or mentioned in the study such as Java, PHP, and C++.
- **Variability model** refers to the variability model used or mentioned in the study such as the feature model, OVM, and goals model.

1.5.1.1 Aspect-oriented programming (AOP)

Aspect-oriented programming (AOP) is an approach that aims to modularize the crosscutting concerns of both software product lines and single-systems. These concerns are widely-scoped properties and usually crosscut several modules in the software system. Aspects are the abstractions used to encapsulate otherwise crosscutting concerns. An example of a crosscutting concern is "logging", which is frequently used in distributed applications to aid debugging by tracing method calls [S34]. Therefore, the use of aspects relies on three major mechanisms to modularize and vary crosscutting concerns: (i) join points are the identifiable execution points, for example, method calls or object attribute assignments; (ii) a pointcut is a predicate over dynamic join points, meaning that given a certain dynamic join point, a pointcut can either match this join point or not (at runtime); and (iii) an advice is a new behavior that extends, refines or replaces the computation at

selected join points [S35]. AspectJ is an example of an aspect-oriented programming language; in AspectJ, a call to a method, a method execution, or an assignment to a variable are examples of join points. For instance, using AspectJ, it is possible to intercept a call to an interface method in order to check whether any of the parameters are null [S35].

A summary of research focusing on SPL and AOP is presented in Table 1-6. This table details important descriptions about the main papers in this field, the SPLE stage covered, the type of evaluation, the kind of examples presented, and the tools, the software languages and the variability model used.

Table 1-6: Summary of SPL and AOP

Ref	SPLE stage covered	Evaluation	Specific Example	Tool Support	Soft. lang.	Variability model
S15	Architecture	Running example	Public health complaint system	COSMOS∗-VP		Feature model
S18	Evaluation	Running example	Mobile phone company	AspectJ - Ant	Java	
S23	Modeling	Running example	Microwave oven	Uced		Feature model
S24	SPL architecture - Multiple	Running example	Graph algorithms	AspectJ - Extended GenVoca - CoBaCoLa - ReGaL	Java	
S25	Multiple	Running example	Scientific calculator	AspectJ	Java	Feature model
S41	Architecture - Evolution	Running example	Philips TV product line architecture	Koala - INXS - AspectC++ - Aspicere - C4 - WeaveC - AspectC		
S42	Evolution	Running example	BestLap, Mobile Media	AspectJ - CaesarJ	Java	Feature model
S44	Customization - Implementation	Running example	Library management domain	AspectJ - OntoFeature	Java	Feature model
S46	Product derivation			AspectJ - pure::variants - Gears - EMF	Java	Feature model
S59	Architecture	Running example	Library management system	AspectJ		Feature model
S60	Multiple	Running example	Game	AspectJ - Junit - EMF - JET - Gears - Pure::variants - Feature modeling plugin (FMP)	Java	Feature model

S66	Architecture	Running example - Comparison	Terrestrial Digital TV System	AspectualACME - ACME		Feature model
S67	Implementation - Multiple	Running example	Weather station - Home automation system	AspectJ - CaesarJ - OSGi - CAM-DAOP - Ecore - openArchitectureWare	Java	Feature model
S70	Modeling - Implementation	Running example	Microwave system	AspectJ	Java - XML	
S85	Multiple	Running example - Comparison	Bus transportation	AspectJ - Spring framework - Hibernate	Java - XML	Feature model

1.5.1.2 Service-oriented architecture (SOA)

SOA emphasizes building software solution logic in the form of self-contained services that can be reused in multiple systems [S6]. In this approach, the SPL components are implemented in the form of services. An SOA implementation exposes standard interfaces to make services available for authorized service consumers to use in a variety of ways [S16], therefore, services can be replaced or can be reconfigured to adapt to different circumstances [S62]. In addition, an SPL that implements an SOA approach commonly uses Business Process Execution Language (BPEL) or similar languages to support variability. BPEL is an XML-based programming language that can be used to describe the interaction between web services at the message level; in this way, it also describes their composition. One example of a BPEL extension is a language called VxBPEL, which has extra XML elements to support variation points and variants in a BPEL process [S62].

SPL and SOA integration has been widely studied over recent years. A summary of the principal research is presented in Table 1-7. This table contains important descriptions about the main papers in this field, the SPLE stage covered, the type of evaluation, the kind of examples presented, and the tools, the software languages and the variability model used.

Table 1-7: Summary of SPL and SOA

Ref	SPLE stage covered	Evaluation	Specific Example	Tool Support	Software language	Variability model
S4	Multiple	Running example	Mobile learning app		Java - Javascript	Feature model
S5	Implementation	Running example	Office system	JBoss jBPM	Java	Feature model
S6	Multiple	Running example	e-health domain	SoaSPL		Feature model

S8	Architecture - Multiple	Running example - Experiment	Currency exchange	VisualWebC - BPEL4WS	ASP - Java	
S14	Multiple					
S16	Implementation - Evaluation	Comparison	Library	OSGi - Apache Tuscany - JAX-WS	Java	Feature model
S17	Multiple	Running example	e-commerce	Apache ODE - Apache CXF - Eclipse Swordfish - SoaSPLE	Java	Feature model - Multiple View Service Variability Model
S26	Implementation - Product derivation	Industrial report	Aurora – web development enviroment		JSP - HTML - XML - Java	
S62	Modeling - Multiple	Running example	Supply Chain Management System	COVAMOF-VS - BPEL - VxBPEL - ArgoUML	Java - XML	UML profile for architectural variability modeling
S68	Multiple					
S76	Modeling - Implementation	Running example	Online marketplace	BPMN - EPC - YAWL - VxBPEL - COVAMOF - SOMA	Java - .net	Feature model

1.5.1.3 Annotative approaches

Annotative approaches implement SPL components with some form of explicit or implicit annotations, with the prototypical example being the use of `#ifdef` and `#endif` statements to surround the SPL component code. Annotative approaches assemble the variations of all possible configurations within a single artifact, as is the case with the C++ preprocessor and the Java preprocessor Antenna [S48]. During variant derivation, the parts that are not needed within a variant are removed. This is the reason that annotative approaches are well known in their support of fine-grained extensions on statements, parameters, and conditional expressions [S65].

A summary of the principal research on SPL and annotative approaches is presented in Table 1-8. This table contains an important description on the major papers in this field, the SPLE stage covered, the type of evaluation, the kind of examples presented, and the tools, the software languages and the variability model used.

Table 1-8: Summary of SPL and annotative approaches

Ref	SPLE stage covered	Evaluation	Specific Example	Tool Support	Software language	Variability model

S2	Multiple	Running example	Satellite system	pure::variants - Rhapsody - C Compiler	C	Feature model
S20	Architecture - Multiple	Running example	Chat app	xADL - ANTLR	Java	
S36	Multiple	Running example	Mobile games		Java	Feature model
S52	Multiple	Running example	Shopping store	GenArch	Java - XML	Feature model
S61	Configuration - Multiple	Running example	Software configuration management	FeaturePlugin - Fujaba - MODPLFeaturePlugin	Java	Feature model
S72	Requirements - Architecture - Multiple	Industrial report	Automotive	pure::variants - DOORS - Rhapsody		Feature model
S84	Architecture - Multiple	Running example	Chat app	xLineMapper - Eclipse JET - ANTLR - ArchJava - Archface	Java	PLA model

1.5.1.4 Feature-oriented programming (FOP) and delta-oriented programming (DOP)

Feature-oriented programming (FOP) has been used to implement SPLs by composing feature modules. To obtain a product for a feature configuration, feature modules are composed incrementally. In the context of OOP, feature modules can introduce new classes or refine existing ones by adding fields and methods or by overriding existing methods [S37]. Delta-oriented programming (DOP) has been seen as an extension of FOP. In DOP, the implementation of an SPL is divided into a core module and a set of delta modules. The core module comprises a set of classes that implement a complete product for a valid feature configuration. This allows the core module to be developed with well-established single application engineering techniques to ensure its quality. Delta modules specify the changes to be applied to the core module in order to implement other products. A delta module can add or remove classes from product implementation [S37].

A summary of the principal research on SPL, FOP, and DOP is presented in Table 1-9. This table contains an important description of the major papers in this field, the SPLE stage covered, the type of evaluation, the kind of examples presented, and the tools, the software languages and the variability model used. References marked with an asterisk (*) use DOP approaches.

Table 1-9: Summary of SPL, FOP, and DOP

Ref	SPLE stage covered	Evaluation	Specific Example	Tool Support	Software language	Variability model
S27*	Modeling - Multiple	Running example	Bank system	ABS tool - ANTLR - JastAdd - Papyrus	Java - XML	Feature model
S32	Multiple	Running example	Graph spl	FeatureC++	C++	Feature model
S39	Multiple	Running example	Graph spl	rbFeatures - AHEAD - CIDE	Ruby	Feature model
S47	Multiple	Running example	Graph spl – calculator – Expression product line	rbFeatures - FeatureJ	Ruby	Feature model
S56	Multiple	Running example	Map	AHEAD - XAK	JavaScript - XML - SVG	Feature model
S75*	Multiple	Running example	Smart home	SiPL - EMF - SiLift - Simulink		Feature model

1.5.1.5 Other approaches

Peña [S31] discussed the use of agents in SPL. Agent-oriented software engineering is a software engineering paradigm that promises to enable the development of more complex systems than those that can be achieved with current object-oriented approaches using agents and organizations of agents as the main abstractions. A software agent is a piece of software which exhibits the following characteristics: autonomy, reactivity, pro-activity and social ability. The introduction of agents to the industrial world may benefit from the advantages that SPL offers [S31]. Using SPL philosophy, a company will be able to define a core multi-agent system from which concrete products will be derived for each customer.

El-Sharkawy *et al.* [S43] developed a tool called EASy-Producer, an Eclipse extension for efficient software product line development. This tool uses three custom-developed domain-specific languages (DSL): IVML language which is used to define the variability model of the SPL, VIL language which is used to define the relationship between the variabilities and the implementation, and VTL language which supports variability-aware artifact generation.

Another approach that the SMS found is context-oriented programming (COP). COP is an approach that supports the dynamic adaptation of context conditions such as bandwidth availability, presence of WiFi and data connection (Salvaneschi *et al.*, 2012). COP introduces language-level abstractions, such as “layers” that group partial method

definitions, to manage the modularization of adaptations and their dynamic activation during the program's execution.

A summary of other SPL implementation approaches is presented in Table 1-10. This table contains an important description of the major papers in this field, the approach used, the SPLE stage covered, the type of evaluation, the kind of examples presented, and the tools, the software languages and the variability model used.

Table 1-10: Summary of other SPL implementation approaches

Ref	Approach	SPLE stage covered	Evaluation	Specific Example	Tool Support	Soft. Lang.	Var. Model
S31	Agents	Multiple	Running example	Security council's procedure to issue resolutions			Feature model - Goals
S43	IVML, VIL, VTL (custom DSLs)	Multiple	Running example	Elevator simulator	EASy-Producer - Dopler tool - FeatureIDE - pure::variant - FaMa framework - REMiDEMMI - FeatureMapper - AspectJ	Java	IVML

1.5.1.6 Mixed approaches

Sometimes, authors develop studies with mixed SPL implementation approaches. Most of these studies compare two or more approaches [S34, S37, S65, S81]. However, in other cases, they try to take advantage of the benefits of some approaches and use them in combination with other approaches. For example, Parra *et al.* [S19] develop an approach for SPL based on SOA services. Their approach mixes the use of SOA services with annotations. The use of annotations serves to manually indicate which parts of the original artifacts can be transformed into services to be used externally. Andrade *et al.* [S22] propose the use of AOP with annotations. In their work, they use AspectJ as the base tool, which is refined with the use of some Java annotations. They implemented this combination to solve some issues presented in previous work which uses AspectJ-based idioms. Santos *et al.* [S73] present RiPLE-HC, a strategy aimed at blending compositional and annotative approaches to implement variability in JavaScript-based systems. In the annotative part,

they use the classical `#ifdef` and `#endif` statements to support the fine-grained extensions. Other studies include the blending of compositional and annotative approaches (Kästner & Apel, 2008; Walkingshaw & Erwig, 2012; Behringer & Rothkugel, 2016; Horcas *et al.*, 2018). However, all the previous approaches present important limitations such as limited support for a few software languages, the use of if statements or tree structures, poorly detailed coding, and no tool support.

A summary of SPL with mixed implementation strategies is presented in Table 1-11. It includes studies that deal with comparative methods. The table contains an important description of the major papers in this field, the mixed approaches used, the SPLE stage covered, the type of evaluation, the kind of examples presented, and the tools, the software languages and the variability model used.

Table 1-11: Summary of SPL with mixed approaches

Ref	Approaches	SPLE stage covered	Evaluation	Specific example	Tool support	Soft. Lang.	Var. Model
S19	SOA - annotative	Multiple	Running example	Multiple artifacts	Spoon - FeatureIDE	Java	Feature model
S22	AOP - Annotative	Implementation - Binding	Experiment	Company spl	AspectJ	Java	
S29	Agents - AOP	Multiple	Running example	Personal user services	GenArch - FMP - Jadex	Java - XML	Feature model
S34	AOP - OO	Evolution	Comparison	Mobile media			Feature model
S35	AOP - OO - Component-based	Architecture - Multiple	Running example	Mobile media	AspectJ	Java	Feature model
S37	DOP - FOP	Multiple	Comparison	Expression product line - Graph	DeltaJ - Jak - AHEAD	Java	Feature model
S48	Annotative - FOP - DOP	Multiple	Running example	FeatureAMP – GameOfLife - Violet	EMF - Xtext - DeltaJ - Antenna - C++ preprocessor	Java – C++	Feature model
S53	AOP - FOP	Multiple	Running example - Comparison	Three board games	AspectJ - CaesarJ	Java	Feature model
S63	OO - AOP	Implementation	Running example - Comparison	Smart homes		Java	Feature model
S65	FOP - OO - Annotative	Evolution - Multiple	Running example - Comparison	WebStore - Mobile Media	AHEAD - JAK	Java - JSP	Feature model

S73	Annotative - compositional	Multiple	Running example - Experiment	Learning objects	RiPLE-HC - npm - jam - bower - requireJS - FeatureHouse - FeatureIDE	JavaScript	Feature model
S78	SOA - annotative	Multiple	Running example	Company apps	SPLIT - Spoon - EMF - FeatureIDE	Java - XML	Feature model
S79	SOA - annotative	Multiple	Running example	Smart home	LISA	Java	Feature model
S81	AOP - OO	Modeling	Running example - Comparison	Pacemaker product line	Rhapsody		

1.5.1.7 SPL implementation approach not specified

There were 23 studies in which the SPL implementation approach that was used was not specified. For these studies, the same information shown in the above tables was recorded. This information can be found online (Correa, 2018).

1.5.1.8 Example of component implementation with some current approaches

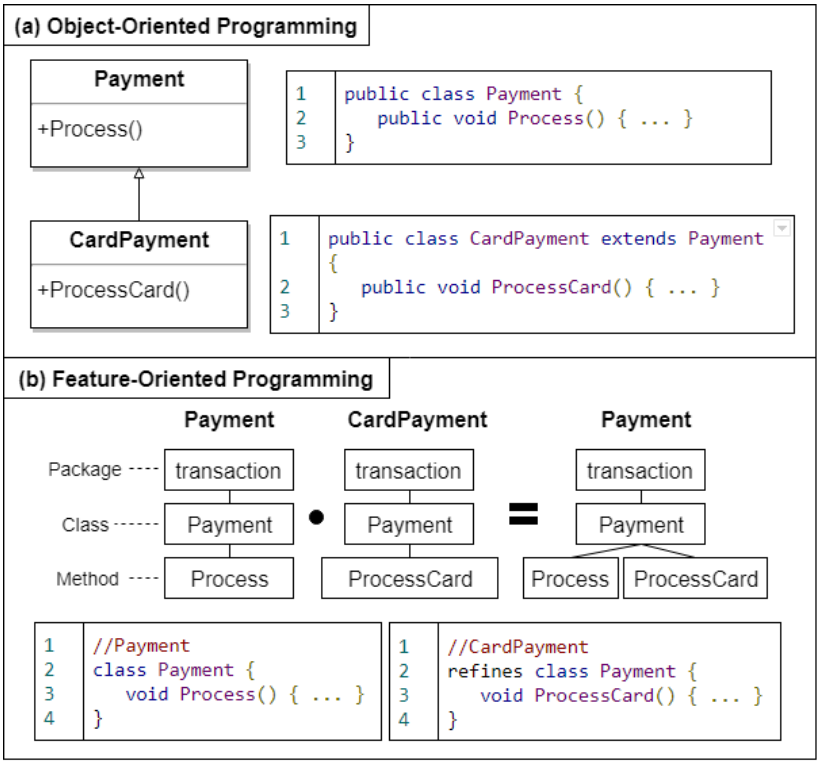
The previous section explained and summarized how different approaches deal with SPL implementation from a theoretical perspective. In order to illustrate how some of those approaches can be used to design and implement SPL components, a practical example of an appliance stores product line was elaborated. In this example, the appliance stores must have the possibility to manage “Payments” (mandatory feature), and those payments can be extended to support “Card Payments” (optional feature). Based on these features, we decided to implement the Payment and Card Payment using six different SPL implementation approaches (OOP, FOP, DOP, COP, AOP and annotative), as shown in Figures 1-8, 1-9, and 1-10. The implemented methods were left blank because the intention is to analyze how the components and their code are divided based on each approach, rather than analyzing the method that was implemented.

OOP and FOP

Figure 1-8-a shows an excerpt of our appliance stores product line represented as a class hierarchy. In OOP, components are implemented with classes. In this example, the “Payment” feature is implemented through the Payment Java class, and the “Card Payment” requirement is implemented through the CardPayment Java class. Extension of

components is an important contribution of OOP to improve reuse by inheritance; however, managing multiple inheritances when the SPL contains hundreds of variants is a very complex task. Figure 1-8-b shows two different options for the implementation of components with FOP. The first option is based on “Superimposition”, which is the process of composing software artifacts by merging their corresponding substructures (cf. Fig. 1-8-b model). The second option is based on refactoring, which has been proposed as a means for improving the internal structure of a system (*i.e.*, the source code) while preserving the external behavior. In this case, a feature module (component) may also contain class and method refinements (cf. Fig. 1-8-b code). The example shows how the CardPayment feature code refines the Payment feature with the addition of the new processCard method. This refinement is carried out in the component integration process. The disadvantage of FOP is that it doesn’t support fine-grained extensions.

Figure 1-8: OOP and FOP implementation of a domain component example

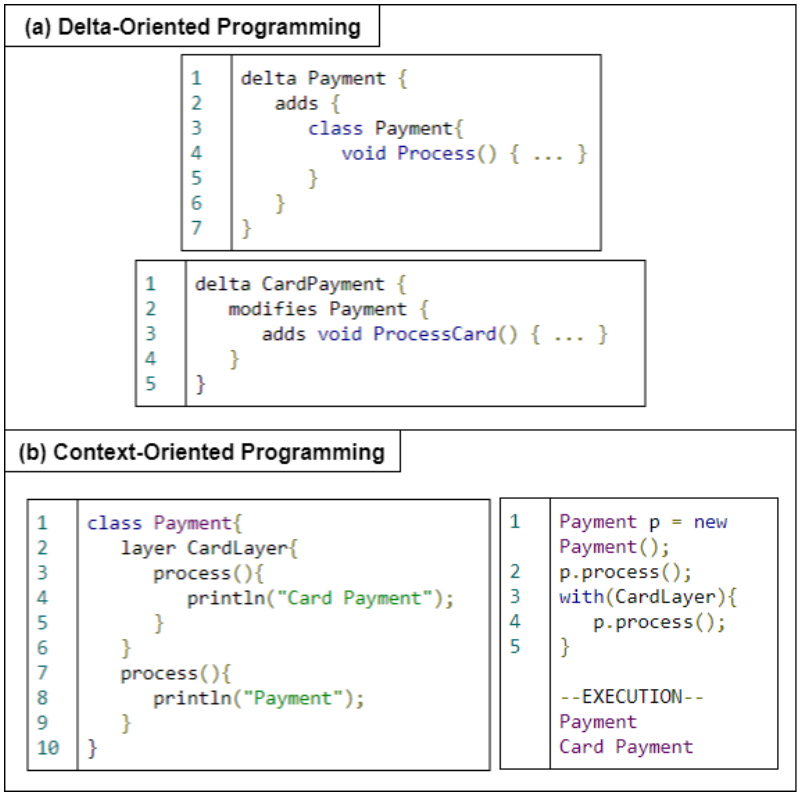


DOP and COP

Figure 1-9-a shows how to implement two components by means of DOP with DeltaJ. Similar to FOP, CardPayment delta modifies the Payment class with the addition of a new method called processCard. Similar to FOP, DOP doesn’t provide fine-grained extensions.

Figure 1-9-b shows the component implementation by means of COP. In our example, the Payment class is extended with a CardLayer, this layer defines a process method, which is only executed when the CardLayer is activated. However, the layer activation mechanism can be quite complex and could tend to make source code complex and difficult to maintain.

Figure 1-9: DOP and COP implementation of a domain component example

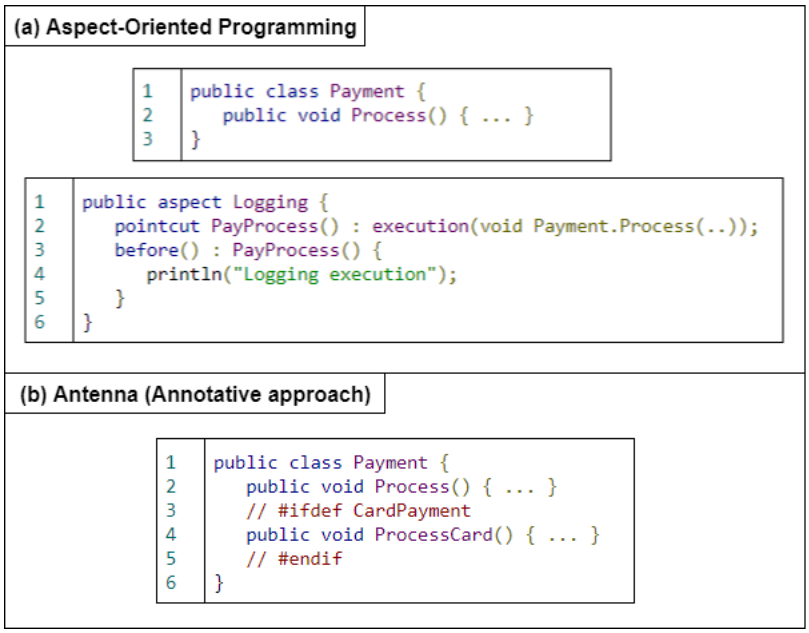


AOP and Antenna (annotative)

Figure 1-10-a shows the component implementation by means of AOP. AOP is a solution for crosscutting concerns, so instead of implementing the CardPayment feature, we implemented a Logging feature which is a crosscutting concern. There, we defined a Logging aspect with a pointcut and advice. The pointcut was called PayProcess which was linked to the Payment Process method. The advice was linked to the previous pointcut and it was defined to be executed before the pointcut execution. This means, that before the Payment Process method is executed, the Logging aspect will execute its advice. The disadvantage of AOP is that it only supports variability for crosscutting concerns. Figure 1-10-b shows how to implement SPL components by means of Antenna (an annotative approach). In this case, the optional feature CardPayment is surrounded by Java

comments. During the product derivation, the parts that are not needed within a file are removed. The disadvantage of annotative approaches is that the SPL components contain all the possible variations inside them, which makes them difficult to maintain and evolve.

Figure 1-10: AOP and Antenna (annotative) implementation of a domain component example



1.5.2 Results

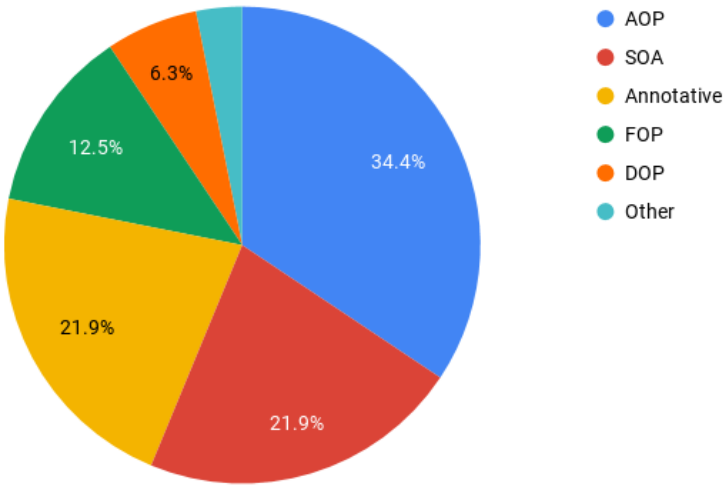
Following, we show the results for answering SRQ4 to SRQ8.

SRQ4: What approaches to implement the SPL components are used?

In the previous section we discussed the studies found about SPL implementation, we also summarized those studies and their contributions. Therefore, we developed the Figures 1-8, 1-9 and 1-10 in which we showed how the SPL components are implemented with some of the previous SPL implementations approaches. Figure 1-11 summarizes the SMS findings of SRQ4. Most of the studies that specified SPL implementation approaches focused on AOP (34%), followed by SOA (22%), annotative approaches (22%), FOP (13%), and DOP (6%). In addition, we found some studies in which there was a mix of approaches to support the SPL component implementation, or in which the authors created some comparisons (see Table 1-11).

Nevertheless, we discussed in Introduction the issues with the previous approaches. For example, some of them support only coarse-grained extensions, are attached to a specific software language and include the code variations inside the reusable component files which increases the complexity, among others.

Figure 1-11: Result of SPL component implementation approaches



SRQ5: What are the available tools that automate SPL implementation?

The first analysis of the SPL tools was to focus on the integrated development environments (IDE). SPL projects are commonly developed inside a specific IDE. We found that the most used or discussed IDE was Eclipse (91%), followed by Visual Studio (9%). There were other development environments such as ArchStudio or pure::variants, but they were developed based on the Eclipse platform.

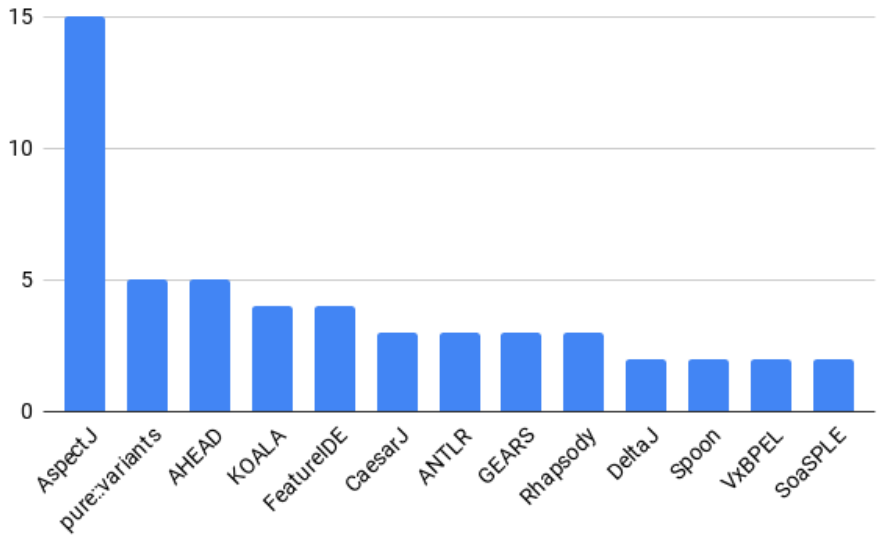
The second analysis was the use of tools for the implementation of an SPL. Most of those tools support a specific SPL implementation approach. Figure 1-12 summarizes the tool results and below we describe the most used or discussed tools based on the SPL implementation approaches:

- **AOP:** AspectJ is an AOP extension for the Java programming language. Currently, AspectJ is the most consolidated AOP language [S42]. CaesarJ is an aspect-oriented language which unifies aspects, classes and packages in a single powerful construct that helps to solve a set of different problems of both aspect-oriented and component-oriented programming.

- **SOA:** VxBPEL is proposed as an extension of Business Process Execution Language (BPEL) for to the process description and definition. VxBPEL allows for run-time variability and variability management in Web service-based systems. Variability information is defined in-line with the process definition [S76]. SoaSPLE is a generic conceptual framework that can be built on top of available Object Management Group (OMG) and standard modeling languages such as UML, BPMN, and SoaML. These languages provide modeling elements that can be used in depicting service views such as SoaMl's Service Interface elements, BPMN's Business Process elements and UML's Interaction Diagram elements [S6].
- **Annotatives:** pure::variants is a tool developed by pure-systems and is used for modeling features, expressing product variants in terms of features, and generating tailored artifacts. It is based on the well-known Eclipse platform and it can be extended by writing plug-ins in the Java programming language [S72]. Rhapsody is a tool developed by IBM Rational, that supports system engineers with modeling static and dynamic aspects of software systems using UML and SysML. Code generation for different languages is supported as well as the simulation of behavioral models such as state charts [S72]. Spoon is a tool that analyses and transforms Java code using processors. Spoon creates an abstract syntax tree of the code being analyzed and offers the API to navigate through the tree and eventually perform modifications [S78]. GEARS provide an all-in-one development environment for establishing, managing and operating your Feature-based PLE Factory. GEARS explicitly supports the integration of existing (i.e., unchanged) software. This implies that GEARS can deal with software over which no control exists [S38].
- **FOP and DOP:** DeltaJ is a programming language which introduces DOP to Java [S37]. DeltaJ is available as an Eclipse plugin and it is based on the Xtext Framework. AHEAD is an approach to support FOP based on stepwise refinements. The main idea behind AHEAD is that programs are constants and features are added to programs using refinement functions [S65].
- **Other:** FeatureIDE is a set of tools for variability modeling that enables one to create and edit feature diagrams. Furthermore, FeatureIDE provides a configuration tool to create and validate configurations with regard to the constraints defined in the variability model. Using FeatureIDE a developer can create product configurations and validate if such configurations respect the constraints expressed in the

variability model [S78]. Koala [S38] is a component model consisting of an architectural description language (ADL) and tool support. The ADL serves to define interfaces, data types, basic components, and compositions (which are components themselves). The tooling serves to generate products from component compositions. Koala was primarily designed for resource-constrained software and is applied in the consumer electronics domain. ANTLR (Another Tool for Language Recognition) is an open-source project that can automatically generate a code processor from a defined grammar. The generated code processor automatically parses the input source code into a syntax tree, and outputs the code as instructed by the user [S20].

Figure 1-12: Quantity of mentions of some SPL implementation tools

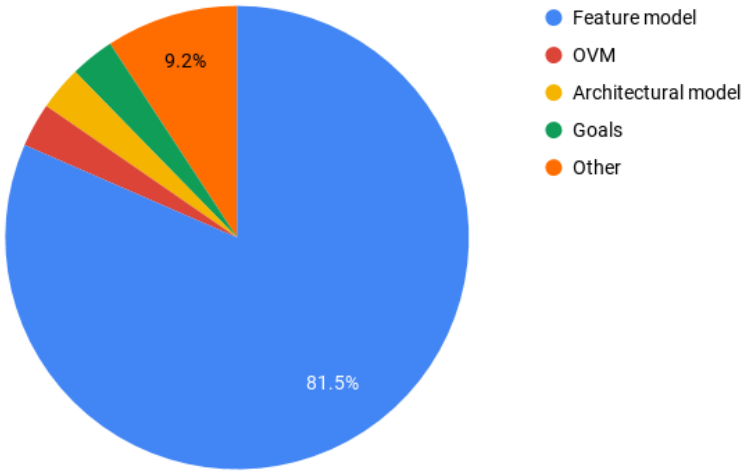


SRQ6: What variability models are most used in an SPL implementation?

Without a question, feature models are the most used and discussed variability models in the SPL implementation literature. Some of the SMS studies specified the use of SPL variability models (see Figure 1-13). In this case, the most used or discussed variability model was feature models (82%). There were other studies in which other variability models were discussed, such as, Orthogonal Variability Model (OVM), architectural models, and goals models. However, they represent a small size of the total studies. During these years,

feature models have become a popular formalism for describing the commonality and variability of SPLs.

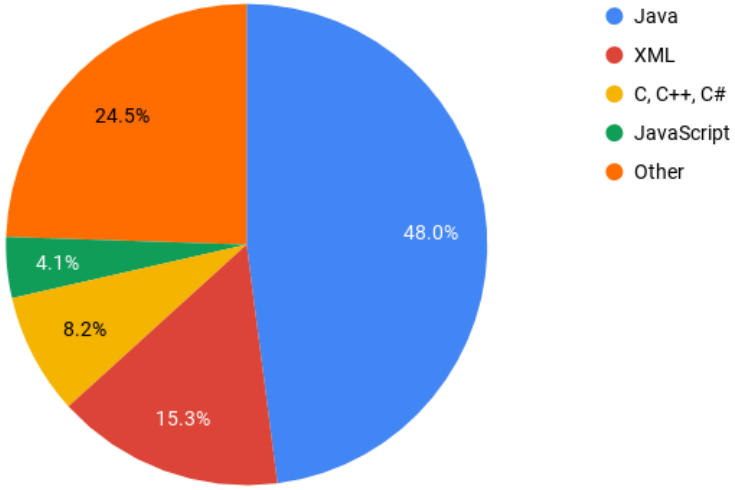
Figure 1-13: Result of variability models



SRQ7: What software languages are most used in the SPL component implementation?

Figure 1-14 summarizes the SMS findings of the software language popularity. In the studies that specify software languages, we found that the most popular was Java (48%). Followed by XML (15%), and C, C++ and C# (8%).

Figure 1-14: Result of software languages

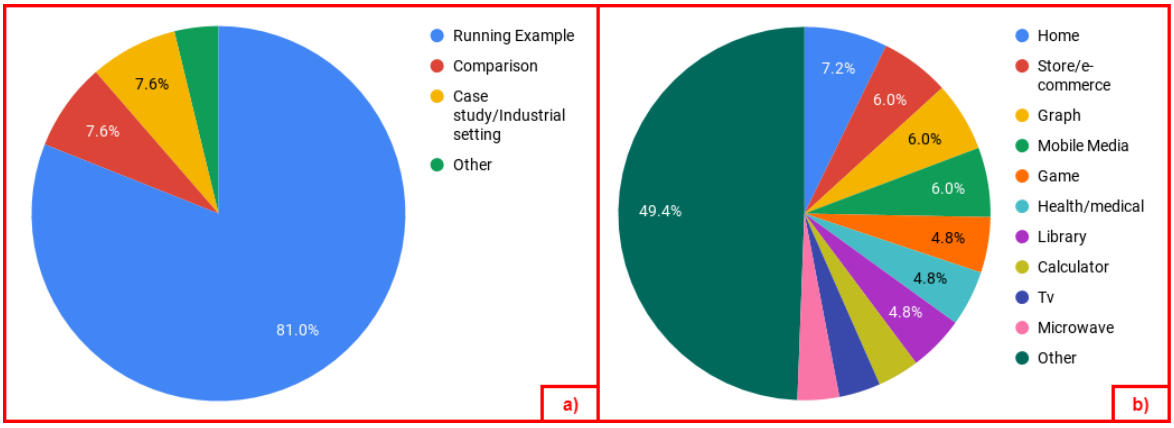


It is not a surprise that Java arises as to the most used software language, this can be due to the fact that the SPL developers are very attached to the use of IDEs such as Eclipse, and many of the authors have proposed and developed tools based on the Eclipse Modeling Framework (EMF). In this framework, developers can specify models with the use of annotated Java, UML, and XML documents, among others. That can be a reason why Java and XML are two of the most used software languages. However, as mentioned in Introduction, many SPL implementation approaches focus on supporting a specific software language or are attached to a specific software language, and software applications use an average of 5 different software languages.

SRQ8: What type of evaluations are most used in studies on SPL implementation?

As we have shown during the development of this chapter, many studies have proposed new mechanisms, concepts, processes, or techniques that improve some stages of the implementation of SPL. Many of those studies also provide a kind of evaluation for their proposals. Figure 1-15-a summarizes SMS evaluations results. Running examples are the most used evaluation technique (81%), which provide a practical way to illustrate the authors' proposals. The use of running examples was followed by comparison studies and case studies both with 8%. It is important to clarify that many studies claimed themselves to use case studies, however, we only recorded case studies for those studies that there were applied in industrial settings. If a study was claimed as a case study but in a non-industrial setting, we recorded them as a running example.

Figure 1-15: Result of evaluations and kind of examples



We also found that most evaluations recorded information such as lines of code (LOC), quantity of classes, quantity of methods, quantity of components, number of product derivations, and number of product configurations, among others.

Finally, we recorded the kind of examples that were presented in the studies' evaluations (see Figure 1-15-b). The results show a wide range of domains in which SPL are applied, which goes from the avionics domain [S50], to games [S36], library management systems [S59], and home automation [S79].

1.6 Resolving questions about research goal G3 – Topics and trends

SRQ9: What are the main topics of the selected studies?

After the SMS development, we have found some main topics in the SPL implementation domain. Some of these topics have been already discussed in Introduction and in Section 1.5; such as SPLE, domain engineering, application engineering, product derivation, variability modeling, compositional and annotative approaches, fine-grained and coarse-grained extensions, component implementation, component assembling, SPL tools, and SPL evaluations. Other main topics are discussed below:

- **MDD – MDE – MDA.** Model-driven development (MDD) and similar areas, such as model-driven engineering (MDE) and model-driven architecture (MDA) improve the way software is developed by capturing key features of a system in models which are developed and refined as the system is created. During the system's lifecycle, models are synchronized, combined and transformed between different levels of abstraction and different viewpoints. In contrast to traditional modeling, models do not only constitute documentation but are processed by automated tools [S67]. Authors and SPL developers have been taken advantage of the MDD characteristics to improve and automatize the implementation of SPL. For example: (i) Alzahmi *et al.* [S6] presented a tool that facilitates the automatic derivation of SOA applications based on MDE as an implementation methodology, (ii) Mefteh *et al.* [S49] developed an approach in which feature models can be built automatically not only from source codes but also from descriptions and uses cases diagrams, and (iii) Mohamed *et al.* [S74] presented a multi-tenant single instance software-as-a-service evolution platform based on Software Product Lines (SPLs) and MDA.

- **PLA – ADL.** Product line architecture (PLA) is an important application of software architecture in the development of a family of software products, or a software product line. It captures architectural commonality and variability among products of the product line [S84]. Architectural description languages (ADLs) can be seemed as an approach for implementing PLA concepts. ADLs typically use architectural styles to define vocabularies of types of components, connectors, properties, and sets of rules that specify how elements of those types may be legally composed in a reusable architectural domain [S66]. PLA and ADL have been also used in conjunction with SPL implementation, Zheng and Cu [S84] presented an approach to implementing product line architecture which combines a code generation and separation pattern with an architecture-based code annotation technique; the Koala tool was designed as a component model consisting of an ADL [S38]; and Barbosa *et al.* [S66] developed PL-AspectualACME which is an extension of the ACME ADL that enriches existing abstractions to express architectural variabilities.
- **DSL.** A domain specific language (DSL) is a formalism for building models which encompasses a meta-model as well as a definition of a concrete syntax that is used to represent the models. The concrete syntax can be textual, graphical or using other means, such as tables, trees or dialogs [S67]. Some authors have developed some DSL to improve the implementation of SPL, such as El-Sharkawy *et al.* [S43] who developed a tool and three custom-made DSLs to support the creation and management of software product line projects, and Pessoa *et al.* [S30] proposed an approach to developing reliable and maintainable DSPLs which uses a DSL to describe reliability goals and adaptability at runtime.
- **DSPL.** Dynamic software product lines (DSPL) have emerged as a promising strategy to develop SPL that incorporate reusable and dynamically reconfigurable artifacts. The central purpose of DSPL is to handle adaptability at runtime through variability management, as well as to maximize the reuse of components [S63].
- **CBSE.** Component-based software engineering (CBSE) focuses on the development and reuse of self-contained software assets in order to achieve better productivity and quality as software systems are composed by previously developed components used (and tested) in other contexts [S28]. CBSE has some similarities with SPL, some SPL developers take some elements from CBSE and incorporate them in SPLE.

- **COTS.** A commercial off-the-shelf (COTS) product or component is one that is used "as-is". COTS components are designed to be easily installed and to interoperate with existing system components. Lago *et al.* [S83] extended a tool to support traceability in product families which allows accommodating both newly developed and COTS components at code level.

SRQ10: What trends have these topics followed over the last years?

For the question resolution about trends, we analyzed the papers presented between 2015 and 2017. We found five main areas in which authors were developing their studies.

- **Dynamic software product lines.** Software availability has become more and more recognized as a quality issue since business transactions and many customer operations have become computerized. DSPL has become a trending topic to support dynamic product reconfiguration and adaptability at runtime [S30,S45,S63], which improve software availability.
- **SMS and SLR studies.** During the last decade, many have authors have developed several proposals in different SPL areas. Recently, some authors have developed multiple SMS and SLR studies trying to provide an overview of these proposals in different SPL areas [S54,S55,S71,S77].
- **Web and mobile systems.** Due to the internet boom, some authors have proposed some studies to apply SPL techniques to web and mobile systems [S9,S73,S80]. This also means that SPL proposals have to evolve to support many mobile and web software languages and frameworks.
- **MDD.** The use of MDD to support some processes inside the SPLE continue being a trending topic [S49, S75]. Research in MDD will allow to automatize the SPLE processes and reduce the manual intervention.
- **PLA.** The evolution of software architectures requires research in the PLA area [S20, S84]. For example, the use of PLA in microservices is a relevant research area.

1.7 Threats to validity

Threats to the validity of the study can be analyzed from the point of view of construct validity, reliability, and internal validity (Wohlin *et al.*, 2000). First, construct validity reflects the extent to which the phenomenon under study really represents what is being

investigated, according to the research questions. The term software product line is well established and hence stable enough to be used as part of the search string. However, for SPL implementation, we consider that this is a controversial term and several authors use different names. That the reason why we divided the SPL implementation term in eight resulting search string, trying to cover as many representative variants as possible (see Table 1-3). Another aspect of the construct validity is the assurance that we find all the papers on the selected topic. We have searched broadly in general publication databases that index the best reputed journals and conference proceedings. The list of different publication media indicates that the width of the search is enough (see Table 1-4). Second, reliability focuses on whether the data are collected, and the analysis is conducted in a way that it can be repeated by other researchers with the same results. We defined the search terms and applied procedures, which may be replicated by others. The non-determinism of some of the databases (Google scholar) is compensated by using more reliable databases (ScienceDirect, Springer, ACM, and IEEE explore). The inclusion/exclusion criteria are related to whether the topic of the field is present in the paper or not. Finally, internal validity is concerned with the analysis of the data. Since the analysis only uses descriptive statistics, the threats are minimal.

1.8 Conclusions

This chapter presented the results of a systematic mapping study on SPL implementation. Including an overview of the processes, methods, and tools used to carry out SPL implementation; and details on the role of the SPL components in the entire process. In total, 88 studies were included in this mapping study from 2000 to March 2017. The SMS included the definition of 10 research questions which were defined and answered. These questions were divided into three categories publication, SPL implementation, and topics and trends. A summary of each category is presented below:

- **Publication.** SPL implementation remains as an interesting field in which many authors from many different countries have been proposing many contributions during the last decade. The most preferred journal to publish this type of articles is Information and Software Technology, and the most preferred conference is the International Software Product Line Conference (SPLC). Brazil leads as the country which has more quantity of publications in this field and has the majority of the top contributions.

- **SPL implementation.** There are several approaches to implement SPL, the most discussed are AOP, SOA, annotative approaches, FOP, and DOP. There are different software tools that support specific approaches or some processes of the SPLE. The most mentioned include AspectJ, pure::variants, AHEAD, KOALA, and FeatureIDE. About variability models and software languages, feature model appears as the most preferred variability model, and Java and XML are the most used software languages. Finally, there are different kinds of evaluations in which the most discussed are running examples, case studies and comparisons.
- **Topics and trends.** Some of the general topics in the SPL implementation domain include SPLE, domain engineering, application engineering, product derivation, variability modeling, compositional and annotative approaches, MDD, PLA, DSL, DSPL, CBSE, and COTS. Therefore, current trends include DSPL, SMS and SLR studies, web and mobile systems, MDD, and PLA.

A. Appendix: List of SMS selected studies

The following contains the list of all the 88 SMS selected studies.

#	**Reference**
S1	Heider, W., Vierhauser, M., Lettner, D., Grunbacher, P.: A case study on the evolution of a component-based product line. In: 2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA), pp. 1–10. IEEE Computer Society, Washington (2012)
S2	Heymans, P., Boucher, Q., Classen, A., Bourdoux, A., & Demonceau, L. (2012). A code tagging approach to software product line development. International Journal On Software Tools For Technology Transfer, 14(5), 553-566.
S3	Shen, L., Peng, X., & Zhao, W. (2009, April). A comprehensive feature-oriented traceability model for software product line development. Australian Software Engineering Conference, 210-219, IEEE.
S4	Falvo, V., Duarte Filho, N. F., Oliveira, E., & Barbosa, E. F. (2014, October). A contribution to the adoption of software product lines in the development of mobile learning applications. Frontiers in Education Conference (FIE), 1-8, IEEE.
S5	Lee, J., Muthig, D., & Naab, M. (2010). A feature-oriented approach for developing reusable product line assets of service-based systems. Journal of Systems and Software, 83(7), 1123-1136.
S6	Alzahmi, S. M., Abu-Matar, M., & Mizouni, R. (2014, April). A Practical Tool for Automating Service Oriented Software Product Lines Derivation. International Symposium on Service Oriented System Engineering (SOSE), 90-97, IEEE.
S7	Deelstra, S., Sinnema, M., & Bosch, J. (2004). A Product Derivation Framework for Software Product Families. Software Product-Family Engineering, 473-484.
S8	Karam, M., Dascalu, S., Safa, H., Santina, R., & Koteich, Z. (2008). A product-line architecture for web service-based visual composition of web applications. Journal of Systems and Software, 81(6), 855-867.
S9	Usman, M., Iqbal, M. Z., & Khan, M. U. (2017). A product-line model-driven engineering approach for generating feature-based mobile applications. Journal of Systems and Software, 123, 1-32.

S10	Go, G., Kang, S., & Ahn, J. (2015, June). A software binding application tool based on the orthogonal variability description language for software product line development. International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 1-8, IEEE.
S11	Lee, J., Kang, S., & Lee, D. (2012, September). A survey on software product line testing. Software Product Line Conference, Vol. 1, 31-40, ACM.
S12	Neto, P. A. D. M. S., do Carmo Machado, I., McGregor, J. D., De Almeida, E. S., & de Lemos Meira, S. R. (2011). A systematic mapping study of software product lines testing. Information and Software Technology, 53(5), 407-423.
S13	Laguna, M. A., & Crespo, Y. (2013). A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Science of Computer Programming, 78(8), 1010-1034.
S14	Khoshnevis, S. (2012, June). An approach to variability management in service-oriented product lines. International Conference on Software Engineering, 1483-1486. IEEE.
S15	Tizzei, L. P., Rubira, C. M., & Lee, J. (2012, September). An aspect-based feature model for architecting component product lines. EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), 85-92. IEEE.
S16	Ribeiro, H. B. G., de Lemos Meira, S. R., de Almeida, E. S., Lucredio, D., Alvaro, A., Alves, V., & Garcia, V. C. (2010, September). An assessment on technologies for implementing core assets in service-oriented product lines. Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 90-99, IEEE.
S17	Abu-Matar, M., & Gomaa, H. (2013, March). An automated framework for variability management of service-oriented software product lines. International Symposium on Service Oriented System Engineering (SOSE), 260-267, IEEE.
S18	Anastasopoulos, M., & Muthig, D. (2004). An Evaluation of Aspect-Oriented Programming as a Product Line Implementation Technology. Software Reuse: Methods, Techniques, And Tools, 141-156.
S19	Parra, C., Joya, D., Giral, L., & Infante, A. (2014, March). An SOA approach for automating software product line adoption. Symposium on Applied Computing, 1231-1238, ACM.
S20	Cu, C., & Zheng, Y. (2016, May). Architecture-centric derivation of products in a software product line. International Workshop on Modeling in Software Engineering, 27-33, IEEE/ACM.
S21	Kim, K., Kim, H., Ahn, M., Seo, M., Chang, Y., & Kang, K. C. (2006, May). ASADAL: a tool system for co-development of software and test environment based on product line engineering. International conference on Software engineering, 783-786, ACM.
S22	Andrade, R., Rebêlo, H., Ribeiro, M., & Borba, P. (2013, September). AspectJ-based idioms for flexible feature binding. Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 59-68, IEEE.
S23	Anthonysamy, P., & Somé, S. S. (2008, March). Aspect-oriented use case modeling for software product lines. AOSD workshop on Early aspects, ACM.

S24	Lesaint, D., & Papamargaritis, G. (2004, June). Aspects and constraints for implementing configurable product-line architectures. Working IEEE/IFIP Conference on Software Architecture (WICSA), 135-144, IEEE.
S25	Lee, K., Botterweck, G., & Thiel, S. (2009, July). Aspectual separation of feature dependencies for flexible feature composition. Computer Software and Applications Conference (COMPSAC), 45-52, IEEE.
S26	Altintas, N. I., Surav, M., Keskin, O., & Cetin, S. (2005, September). Aurora software product line. Turkish Software Architecture Workshop, Ankara.
S27	Muhammad, R., & Setyautami, M. R. A. (2016, October). Automatic model translation to UML from software product lines model using UML profile. International Conference on Advanced Computer Science and Information Systems (ICACSIS), 605-610, IEEE.
S28	Miranda Filho, S., Mariano, H., Kulesza, U., & Batista, T. (2010, September). Automating Software Product Line Development: A Repository-Based Approach. Conference on Software Engineering and Advanced Applications (SEAA), 141-144, IEEE.
S29	Cirilo, E., Nunes, I., Kulesza, U., & Lucena, C. (2012). Automating the product derivation process of multi-agent systems product lines. Journal of Systems and Software, 85(2), 258-276.
S30	Pessoa, L., Fernandes, P., Castro, T., Alves, V., Rodrigues, G. N., & Carvalho, H. (2017). Building reliable and maintainable Dynamic Software Product Lines: An investigation in the Body Sensor Network domain. Information and Software Technology, 86, 54-70.
S31	Peña, J. (2005, September). Can agent oriented software engineering be used to build MASs product lines?. Workshop on Radical Agent Concepts, 98-108, Springer Berlin Heidelberg.
S32	Rosenmüller, M., Siegmund, N., Saake, G., & Apel, S. (2008, October). Code generation to support static and dynamic composition of software product lines. International conference on Generative programming and component engineering, 3-12, ACM.
S33	Mohabbati, B., Asadi, M., Gašević, D., Hatala, M., & Müller, H. A. (2013). Combining service-orientation and software product line engineering: A systematic mapping study. Information and Software Technology, 55(11), 1845-1859.
S34	Abdelmoez, W., Khater, H., & El-shoafy, N. (2012, May). Comparing maintainability evolution of object-oriented and aspect-oriented software product lines. International Conference on Informatics and Systems (INFOS), SE-53 - SE-60, IEEE.
S35	Tizzei, L. P., Dias, M., Rubira, C. M., Garcia, A., & Lee, J. (2011). Components meet aspects: Assessing design stability of a software product line. Information and Software Technology, 53(2), 121-136.
S36	Nascimento, L. M., de Almeida, E. S., & de Lemos Meira, S. R. (2009). Cores assets development in software product lines-towards a practical approach for the mobile game domain. Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS), 124-137.
S37	Schaefer, I., Bettini, L., Bono, V., Damiani, F., & Tanzarella, N. (2010). Delta-Oriented Programming of Software Product Lines. Software Product Lines: Going Beyond, 77-91.
S38	de Jonge, M. (2009). Developing product lines with third-party components. Electronic Notes in Theoretical Computer Science, 238(5), 63-80.

S39	Günther, S., & Sunkle, S. (2010, October). Dynamically adaptable software product lines using Ruby metaprogramming. International Workshop on Feature-Oriented Software Development, 80-87, ACM.
S40	Capilla, R., & Dueñas, J. C. (2005). Evolution and Maintenance of Web Sites: A Product Line Model. Managing Corporate Information Systems Evolution and Maintenance, 255-271.
S41	Tesanovic, A. (2007, March). Evolving embedded product lines: opportunities for aspects. Workshop on Aspects, components, and patterns for infrastructure software, ACM.
S42	Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Castor Filho, F., & Dantas, F. (2008, May). Evolving software product lines with aspects: an empirical study on design stability. International conference on Software engineering, 261-270, ACM/IEEE.
S43	El-Sharkawy, S., Kröher, C., Eichelberger, H., & Schmid, K. (2015, October). Experience from implementing a complex eclipse extension for software product line engineering. Eclipse Technology eXchange, 13-18, ACM.
S44	Peng, X., Shen, L., & Zhao, W. (2008). Feature Implementation Modeling Based Product Derivation in Software Product Line. High Confidence Software Reuse in Large Systems, 142-153.
S45	Amja, A. M., Obaid, A., Mili, H., & Jarir, Z. (2016, November). Feature-Based Adaptation and Its Implementation. International Conference on Collaboration and Internet Computing (CIC), 321-328, IEEE.
S46	Lee, K., Botterweck, G., & Thiel, S. (2009, May). Feature-modeling and aspect-oriented programming: Integration and automation. International Conference on Software Engineering, Artificial Intelligences, Networking and Parallel/Distributed Computing (SNPD), 186-191, IEEE.
S47	Günther, S., & Sunkle, S. (2009, October). Feature-oriented programming with Ruby. International Workshop on Feature-Oriented Software Development, 11-18, ACM.
S48	Seidl, C., Schuster, S., & Schaefer, I. (2017). Generative software product line development using variability-aware design patterns. Computer Languages, Systems & Structures, 48, 89-111.
S49	Mefteh, M., Bouassida, N., & Ben-Abdallah, H. (2015, April). Implementation and evaluation of an approach for extracting feature models from documented UML use case diagrams. Symposium on Applied Computing, 1602-1609, ACM.
S50	Dordowsky, F., Bridges, R., & Tschope, H. (2011, August). Implementing a software product line for a complex avionics system. Software Product Line Conference (SPLC), 241-250, IEEE.
S51	Geertsema, B., & Jansen, S. (2010, August). Increasing software product reusability and variability using active components: a software product line infrastructure. European Conference on Software Architecture: Companion Volume, 336-343, ACM.
S52	Cirilo, E., Kulesza, U., Coelho, R., de Lucena, C., & von Staa, A. (2008). Integrating Component and Product Lines Technologies. High Confidence Software Reuse in Large Systems, 130-141.
S53	Gurgel, A., Dantas, F., Garcia, A., & Sant'Anna, C. (2012, July). Integrating Software Product Lines: A Study of Reuse versus Stability. Computer Software and Applications Conference (COMPSAC), 89-98, IEEE.

S54	Afzal, U., Mahmood, T., & Shaikh, Z. (2016). Intelligent software product line configurations: A literature review. Computer Standards & Interfaces, 48, 30-48.
S55	Méndez-Acuna, D., Galindo, J. A., Degueule, T., Combemale, B., & Baudry, B. (2016). Leveraging Software Product Lines Engineering in the development of external DSLs: A systematic literature review. Computer Languages, Systems & Structures, 46, 206-235.
S56	Freeman, G., Batory, D., & Lavender, G. (2008). Lifting Transformational Models of Product Lines: A Case Study. Theory And Practice Of Model Transformations, 16-30.
S57	McRitchie, I., Brown, T., & Spence, I. (2004). Managing Component Variability within Embedded Software Product Lines via Transformational Code Generation. Software Product-Family Engineering, 98-110.
S58	Thao, C. (2012, June). Managing evolution of software product line. International Conference on Software Engineering (ICSE), 1619-1621, IEEE.
S59	Zhang, J., Cai, X., & Liu, G. (2008, December). Mapping features to architectural components in aspect-oriented software product lines. International Conference on Computer Science and Software Engineering, vol. 2, 94-97, IEEE.
S60	Kulesza, U., Alves, V., Garcia, A., Neto, A., Cirilo, E., de Lucena, C., & Borba, P. (2007). Mapping Features to Aspects: A Model-Based Generative Approach. Early Aspects: Current Challenges and Future Directions, 155-174.
S61	Buchmann, T., Dotor, A., & Westfechtel, B. (2013). MOD2-SCM: A model-driven product line for software configuration management systems. Information and Software Technology, 55(3), 630-650.
S62	Sun, C. A., Rossing, R., Sinnema, M., Bulanov, P., & Aiello, M. (2010). Modeling and managing the variability of Web service-based systems. Journal of Systems and Software, 83(3), 502-516.
S63	Carvalho, M. L. L., Gomes, G. S. D. S., Da Silva, M. L. G., Machado, I. D. C., & de Almeida, E. S. (2016, September). On the Implementation of Dynamic Software Product Lines: A Preliminary Study. Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), 21-30, IEEE.
S64	Barros, F. J. (2013, April). On the representation of product lines using pluggable software units: results from an exploratory study. Symposium on Theory of Modeling & Simulation-DEVS Integrative M&S Symposium, Society for Computer Simulation International.
S65	Ferreira, G. C. S., Gaia, F. N., Figueiredo, E., & de Almeida Maia, M. (2014). On the use of feature-oriented programming for evolving software product lines—A comparative study. Science of Computer Programming, 93, 65-85.
S66	Adachi Barbosa, E., Batista, T., Garcia, A., & Silva, E. (2011). PL-AspectualACME: An Aspect-Oriented Architectural Description Language for Software Product Lines. Software Architecture, 139-146.
S67	Voelter, M., & Groher, I. (2007, September). Product line implementation using aspect-oriented and model-driven software development. Software Product Line Conference (SPLC), 233-242, IEEE.

S68	Capilla, R., & Topaloglu, N. Y. (2005, September). Product lines for supporting the composition and evolution of service oriented applications. International Workshop on Principles of Software Evolution, 53-56, IEEE.
S69	Caporuscio, M., Muccini, H., Pelliccione, P., & Di Nisio, E. (2006). Rapid System Development Via Product Line Architecture Implementation. Rapid Integration Of Software Engineering Techniques, 18-33.
S70	Heo, S. H., & Choi, E. M. (2006, August). Representation of variability in software product line using aspect-oriented programming. International Conference on Software Engineering Research, Management and Applications, 66-73, IEEE.
S71	Montalvillo, L., & Díaz, O. (2016). Requirement-driven evolution in software product lines: A systematic mapping study. Journal of Systems and Software, 122, 110-143.
S72	Derakhshanmanesh, M., Fox, J., & Ebert, J. (2014). Requirements-driven incremental adoption of variability management techniques and tools: an industrial experience report. Requirements Engineering, 19(4), 333-354.
S73	Santos, A. R., do Carmo Machado, I., & de Almeida, E. S. (2016, September). RiPLE-HC: Javascript systems meets SPL composition. Systems and Software Product Line Conference, 154-163, ACM.
S74	Mohamed, F., Abu-Matar, M., Mizouni, R., Al-Qutayri, M., & Al Mahmoud, Z. (2014, December). SaaS Dynamic Evolution Based on Model-Driven Software Product Lines. 6th International Conference on Cloud Computing Technology and Science (CloudCom), 292-299, IEEE.
S75	Pietsch, C., Kehrer, T., Kelter, U., Reuling, D., & Ohrndorf, M. (2015, November). SiPL--A Delta-Based Modeling Framework for Software Product Line Engineering. International Conference on Automated Software Engineering (ASE), 852-857, IEEE/ACM.
S76	Mohabbati, B., Asadi, M., Gašević, D., & Lee, J. (2014). Software Product Line Engineering to Develop Variant-Rich Web Services. Web Services Foundations, 535-562.
S77	Vale, T., de Almeida, E. S., Alves, V., Kulesza, U., Niu, N., & de Lima, R. (2017). Software product lines traceability: A systematic mapping study. Information and Software Technology, 84, 1-18.
S78	Parra, C., & Joya, D. (2015). SPLIT: an automated approach for enterprise product line adoption through SOA. Journal of Internet Services and Information Security (JISIS), 5(1), 29-52.
S79	Groher, I., & Weinreich, R. (2013, January). Supporting variability management in architecture design and implementation. Hawaii International Conference on System Sciences (HICSS), 4995-5004, IEEE.
S80	Carromeu, C., Paiva, D., & Cagnin, M. (2015). The Evolution from a Web SPL of the e-Gov Domain to the Mobile Paradigm. Computational Science And Its Applications -- ICCSA 2015, 217-231.
S81	Liu, J. J., Lutz, R. R., & Rajan, H. (2006, October). The role of aspects in modeling product line variabilities. Workshop on Aspect-oriented Product Line Engineering (AOPLE), 32-39.
S82	dos Santos Rocha, R., & Fantinato, M. (2013). The use of software product lines for business process management: A systematic literature review. Information and Software Technology, 55(8), 1355-1373.

S83	Lago, P., Niemela, E., & Van Vliet, H. (2004, March). Tool support for traceable product evolution. Conference on Software Maintenance and Reengineering (CSMR), 261-269, IEEE.
S84	Zheng, Y., & Cu, C. (2016, May). Towards implementing product line architecture. Workshop on Bringing Architectural Design Thinking into Developers' Daily Activities, 5-10, IEEE/ACM.
S85	de Moraes, A. L., Brito, R. D. C., Junior, A. C. C., Ramos, M. C., Colanzi, T. E., Gimenes, I. M. D. S., & Masiero, P. C. (2010, November). Using aspects and the spring framework to implement variabilities in a software product line. International Conference of the Chilean Computer Science Society (SCCC), 71-80, IEEE.
S86	Hartmann, H., Keren, M., Matsinger, A., Rubin, J., Trew, T., & Yatzkar-Haham, T. (2013). Using MDA for integration of heterogeneous components in software supply chains. Science of Computer Programming, 78(12), 2313-2330.
S87	Mazo, R., Assar, S., Salinesi, C., & Hassen, N. B. (2014). Using Software Product Line to improve ERP Engineering: literature review and analysis. Latin American Journal of Computing Faculty of Systems Engineering National Polytechnic School Quito-Ecuador, 1(1), pp. 10.
S88	H+A1:B89umblet, M., Tran, D. V., Weber, J. H., & Cleve, A. (2016, May). Variability management in database applications. Workshop on Variability and Complexity in Software Design, 21-27, ACM.