Science of Computer Programming 78 (2013) 1010–1034

Contents lists available at SciVerse ScienceDirect

Science of Computer Programming

journal homepage: www.elsevier.com/locate/scico

A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring

Miguel A. Laguna *, Yania Crespo

GIRO research group, University of Valladolid, Spain

ARTICLE INFO

Article history:
Received 11 April 2011
Received in revised form 10 May 2012
Accepted 14 May 2012
Available online 22 May 2012

Keywords:
Software product line
Evolution
Reengineering
Legacy system
Refactoring

ABSTRACT

Software product lines (SPLs) are used in industry to develop families of similar software systems. Legacy systems, either highly configurable or with a story of versions and local variations, are potential candidates for reconfiguration as SPLs using reengineering techniques. Existing SPLs can also be restructured using specific refactorings to improve their internal quality. Although many contributions (including industrial experiences) can be found in the literature, we lack a global vision covering the whole life cycle of an evolving product line. This study aims to survey existing research on the reengineering of legacy systems into SPLs and the refactoring of existing SPLs in order to identify proven approaches and pending challenges for future research in both subfields. We launched a systematic mapping study to find as much literature as possible, covering the diverse terms involved in the search string (restructuring, refactoring, reengineering, etc. always connected with SPLs) and filtering the papers using relevance criteria. The 74 papers selected were classified with respect to several dimensions: main focus, research and contribution type, academic or industrial validation if included, etc. We classified the research approaches and analyzed their feasibility for use in industry. The results of the study indicate that the initial works focused on the adaptation of generic reengineering processes to SPL extraction. Starting from that foundation, several trends have been detected in recent research: the integrated or guided reengineering of (typically object-oriented) legacy code and requirements; specific aspect-oriented or feature-oriented refactoring into SPLs, and more recently, refactoring for the evolution of existing product lines. A majority of papers include academic or industrial case studies, though only a few are based on quantitative data. The degree of maturity of both subfields is different: Industry examples for the reengineering of the legacy system subfield are abundant, although more evaluation research is needed to provide better evidence for adoption in industry. Product line evolution through refactoring is an emerging topic with some pending challenges. Although it has recently received some attention, the theoretical foundation is rather limited in this subfield and should be addressed in the near future. To sum up, the main contributions of this work are the classification of research approaches as well as the analysis of remaining challenges, open issues, and research opportunities.

1. Introduction

Software Product Lines (SPLs) are successfully used in industry to achieve a disciplined development of families of software systems [1]. SPLs combine systematic development and the reuse of coarse-grained components that include the

* Corresponding author.
E-mail addresses: mlaguna@infor.uva.es (M.A. Laguna), yania@infor.uva.es (Y. Crespo).

0167-6423/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.scico.2012.05.003

common and variable parts of the product line [2,3]. Many legacy systems, either highly configurable or with a long history of versions and local variations, are susceptible to reengineering as SPLs [4]. The catalog of techniques for evolving legacy software to become an SPL can be really extensive. Some are closely related to generic reengineering approaches. Others can be seen as model and code transformation techniques. In particular, refactoring techniques can be used to reconfigure the internal details of the legacy systems. However, the quality of an SPL resulting from the reengineering of legacy systems is often weak, especially regarding code and architecture quality, which is why refactoring techniques are often applied too. Existing SPLs can also be restructured using specific refactorings to improve their internal quality. Diverse academic proposals and industrial experiences addressing these topics are present in the literature, but a global vision that covers the whole life cycle of an evolving product line is missing. Some of our previous work was focused on SPLs [5,6] and refactoring [7,8], as separate topics. We were interested in the synergy of both fields to adapt existing techniques or to develop specific new ones to facilitate the conversion of (families of) legacy systems into an SPL, and/or evolve existing SPLs following a quality-driven process. With this aim, the need for a systematic search of the literature arises. This would help to clarify the envisioned possibilities and to establish a good foundation for our main goal, revealing remaining challenges, trends and open issues.

A systematic literature review uses a rigorous procedure for searching and analyzing previous research papers and has been adopted in Evidence-Based Software Engineering [9]. A systematic review tries to obtain all the relevant papers (the primary studies) for a specific question or area of interest, and to use them to summarize, assess and interpret the relevance of all evidence related to that question, minimizing possible bias. It is aimed at providing knowledge about when, how, and in what context technologies, processes, methods or tools are more appropriate for software engineering practices [10]. The phases of the review include planning (the research questions and the review protocol are defined), conduction (primary studies are identified, selected, and evaluated; for each selected study, data are extracted, analyzed and synthesized), and reporting (a final report is presented).

However, when a research area is evolving and there is an increasing number of results available, the research questions can be diverse and a previous effort of clarification and classification of the new approaches is required. To summarize, and to obtain an overview of the field of interest, mapping studies provide a comprehensive evaluation of research using a research strategy similar to systematic reviews [11]. A systematic mapping study allows the results that have been previously published to be categorized, and often gives a visual map of its results. Kitchenham et al. remark that mapping studies use the same basic methodology as systematic reviews but aim to identify and classify all research related to a broader software engineering topic, rather than answering questions about the relative merits of competing technologies [12]. They conclude that mapping studies provide a foundation to assist more focused research efforts.

We conducted our systematic mapping study in the software product line domain and aimed at identifying relevant primary studies related to the reengineering of legacy systems into SPLs or to the refactoring of these SPLs to improve their quality. It was initially conducted from October to December, 2010 and was carried out by two people (SPL and refactoring specialists). Second and third revisions were carried out at the end of 2011, and in April 2012, extending the search string with the "product family" related terms and incorporating 2011 studies. Adapting the suggestions proposed by Petersen et al. [11], the defined protocol included the following steps:

1. Definition of the research questions, the scope and strategy of the search, and the inclusion and exclusion criteria.
2. Conduction of the search for primary studies, using the established protocol and relevance and quality criteria (context, clear objectives, description of the research method) which enable the assessment of the quality of primary studies addressing bias and validity.
3. Screening of the papers, based on inclusion/exclusion criteria, and examining references to ensure the presence of as many articles as possible.
4. Classification of the papers, assigning diverse tags, following multifaceted criteria.
5. Data extraction and aggregation in several tables to have an overview of the field.

The rest of the paper is structured as follows. Some background on SPLs, refactoring and related terminology is introduced in Section 2. The following sections present how the above mentioned steps have been carried out in our systematic mapping. Section 3 describes how the mapping study methodology has been planned (step 1). Section 4 presents the details of the search and classification of the papers, compiling some statistics on the primary studies (steps 2, 3 and 4). Section 5 (legacy system reengineering) and 6 (product line refactoring) are organized into several blocks and tables and analyze the results of that classification (step 5). A discussion of the due and remaining challenges extracted from the mapping study in response to the research questions are also analyzed in both Sections 5 and 6. Finally, the conclusions and a summary of the state of the art and research opportunities are provided in Section 7.

2. Background

Opdyke [13] stamped the term refactoring and defined what can be considered as the first refactoring catalog. Fowler et al. (Opdyke and Beck as contributors) published the first book on refactoring. According to Fowler's definition, refactoring is "a technique for restructuring an existing body of code, altering its internal structure without changing its external behavior" [14]. The last decade has witnessed a large number of contributions in software refactoring. Refactoring operations were defined, automated and integrated into software development processes. Refactoring opportunities were automatically

detected as well. Closely-related to this matter, design smells were precisely defined and automatically detected [15]. Our research is broadly aimed at improving the automation of complex refactoring processes. Complex refactoring processes can be targeted to many different goals, such as introducing or removing design patterns as presented by Kerievsky [16], improving certain software quality factors, or applying a big refactoring [14]. We are interested in exploring the complex refactoring processes aimed at improving the quality of an existing SPL. This is one of the starting points of the present work.

The well-established definition of refactoring can be used, in particular, to improve the quality of an existing SPL. However, when the SPL is built starting from legacy systems, the term refactoring is too restrictive because the behavior of each individual, common or variable part can be slightly different from the original system. Software restructuring was defined as "the process of re-organizing the logical structure of existing software systems in order to improve particular quality attributes of software products" [17]. Therefore, software refactoring processes can be seen as a special type of software restructuring, particularly constrained to behavior preservation.

Software reengineering is a related concept with a long history and some controversy. Software reengineering (sometimes spelled as re-engineering) was originally defined as "the examination and alteration of a system to reconstitute it in a new form" [18]. The primary goal was to attain new levels of efficiency of the existing assets, without recurring to the development of new systems. Typically, reengineering is carried out to re-structure a legacy system into a new system with better functionalities and probably adapted to new hardware and software constraints. The concept of reengineering is well-suited to the process of obtaining an SPL from legacy systems.

There are other related terms such as migration and evolution. Migration is a term used more in the context of changing languages, databases, or platforms, often using automated converters as a way of transforming legacy systems. Finally, the generic term evolution refers to "a process of progressive change in the attributes of the evolving entity or that of one or more of its constituent elements" [19]. Usually, it refers to software changes over time, dictated by the change of the initial requirements [20]. Breivold et al. [21] have recently published a systematic literature review of software architecture evolution research, focused on analyzing and improving software evolvability at the architectural level. They specifically searched studies on software architecture properties (evolvability, maintainability, adaptability, etc.). Their field of study is more restricted (only architecture level properties) and focused on conventional products instead of software product lines).

Given the different goals of migration and (temporal) evolution concepts, we decided to focus our study on the refactoring/restructuring/reengineering terms to enclose these related techniques in a global view, avoiding the constraint that implies the use of the refactoring term alone. The strings used for searching the primary studies must take these variants into account, always in the context of SPLs. Independently of the terms used by the original authors, we consider SPL building from legacy systems as *reengineering* (an SPL is essentially different from the sum of previous products), using the term *refactoring* in the more restrictive sense, close to the original definition, of evolving an existing SPL or part of it, and improving its quality without changing its behavior. As a matter of convenience, we will refer to both research subfields jointly with the abbreviated expression *SPL oriented evolution.*

A software product line is "a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [2]. Several large companies (including Nokia, Philips, etc.) and academic institutions have conducted research projects and developed industrial SPLs [1]. In Academia, the best known are the Product Line Practice Initiative at the *Software Engineering Institute* (SEI) of the Carnegie Mellon University and the set of projects launched in the European *Fraunhofer Institute for Experimental Software Engineering* (IESE).

The SEI compiled practice areas for software product line engineering [2]. One of these practice areas was the reengineering of legacy systems, introducing the methods MAP [22] and OAR [23]. Both methods described the selection process of existing components for integration in a product line. However, they did not provide technical details on code analysis or tool support.

The IESE [24] developed another product line framework, called PulSE, which used a product development cycle and included the necessary reuse infrastructure. An integral part was RE-PLACE [25], a process to integrate existing systems into an SPL. This approach used wrapping to mitigate interface discrepancies between the SPL and the legacy components without changing the latter. RE-PLACE is a pure process description and does not have concrete techniques to support it.

These pioneer works have been outdated and replaced by more focused research with diverse orientations. A systematic study is needed to reflect the progress of this knowledge domain during the last few years.

3. Systematic mapping study planning

This section describes how the mapping study methodology has been planned, including the research questions, the scope and strategy of the search, and the inclusion, exclusion, and classification criteria.

3.1. Research questions

The goal of this study is to get an overview of existing research on *SPL oriented evolution*, from the initial phases (reengineering of the legacy systems into a new SPL) to the latest (refactoring of the SPL to improve its quality). A further

Table 1
Selected databases.

Source	Location
Science Direct	www.sciencedirect.com
Scopus	http://www.scopus.com
Web of Science	www.isiknowledge.com
IEEE Xplore	www.ieeexplore.ieee.org
ACM Digital Library	www.portal.acm.org
Springer	http://www.springerlink.com
Google Scholar[a]	scholar.google.com
Citeseerx[a]	http://citeseerx.ist.psu.edu

[a] Secondary sources.

interest is to assess the maturity of the diverse research efforts. Approaches are evaluated by means of well-founded empirical studies to increase the credibility of the research [26]. The availability of tools (allowing automated refactoring, for example) or industry experiences indicate better support for a particular method against others. Thus, we must highlight the techniques that are being used (or ready to be used) in industry and the aspects that need more study, offering research opportunities. The overall goal is divided into the following research questions:

- RQ1 - What approaches have been proposed on SPL oriented evolution and what is their focus and origin? This question involves the topics themselves but also the validation degree of each generic approach and can be completed with two more detailed questions:
  - RQ1.1 - Which methods or techniques have been investigated and to what extent? These can be the different technical or managerial aspects, and can involve different kinds of artifacts: requirements, variability models, generic architecture, etc.
  - RQ1.2 - What is the maturity level of the approach? Which tools are already available and which ones are currently used in industry? What types of validation studies are represented and to what extent?
- RQ2 - Which challenges for SPL oriented evolution have been identified? Challenges can be identified as a byproduct of this study. Our aim is to get an overview of the challenges to identify the relevance of future research.

3.2. Search strategy

Considering the research questions, we have identified a set of search keywords. Thus, the keyword "software product line(s)" must be included to establish the general context, though "software" can be safely removed to avoid discarding some papers without any danger of increasing the number of hits. In a further revision, *"product family"* related keywords were added as, in some contexts, these are the preferred terms. The second keyword must be "legacy" or a word that indicates some type of restructuration process. We have selected "refactoring", "reengineering" and "restructuring", covering the different terms that we have found in a previous study of some well-known sources, specifically the SEI and IESE institutes mentioned above. It is worth highlighting that the keywords chosen should be simple enough to bring many results and, at the same time, rigorous enough to cover only the desired research topic. We used the Boolean OR operators to link the main terms and their synonyms. Finally, we combined these terms using the Boolean AND operator. Thus, the final search string was:

("product line" OR "product lines" OR "product-line" OR "product-lines" OR "product family" OR "product families" OR "product-family" OR "product-families") AND (legacy OR refactoring OR reengineering OR restructuring OR re-engineering OR re-structuring).

In addition to the research questions and search strategy, we established the search sources used to find the primary studies. The criteria used to select the sources follow the proposal discussed in [27]: content update (publications are regularly updated); availability (full text of the papers is available); quality of results (accuracy of the results returned by the search); and versatility export (since much information is returned through the search, a mechanism to export the results is required). Thus, the selected databases for our systematic mapping study are shown in the first six rows of Table 1. According to Dyba et al. [28], these databases are efficient to conduct systematic studies in the context of software engineering.

After a first consolidation of the results, in a second phase we considered other databases (Google Scholar and Citeseerx) to try to find additional results that can offer useful material irrespective of its relevance (for example a technical report providing a set of citations that could be used to check the set of already encountered papers). The screening of the cited references section inside the selected papers can be used to complete the set of papers with works otherwise neglected due to the words used in their title or keywords.

3.3. Inclusion/exclusion criteria

Another important element of the systematic review planning is to define the Inclusion Criteria and Exclusion Criteria. These criteria make it possible to include primary studies that are relevant to answering the research questions and

Table 2
Inclusion and exclusion criteria.

Inclusion criteria
• English peer-reviewed articles in conferences or journals published until Dec. 2011 • Articles that focus on software product lines • Articles that provide some type of evolution of existing software artifacts, included among the terms selected • Context, objectives, and research method are reasonably present
Exclusion criteria
• Articles that are not related to software product lines • Articles that do not imply evolution of any software artifacts • Non-peer reviewed publications • Articles that are not written in English • Context, objectives, or research method are manifestly missing

exclude studies that do not answer them. We have summarized the inclusion and exclusion criteria of our systematic mapping study in Table 2. The criteria used to select papers that permit the research questions to be answered are:

- The primary focus of the paper is SPLs (including product families).
- The paper presents experiences involving some type of evolution of existing software artifacts, including possibly other product lines or existing software families.
- The target of the evolution could vary. The evolution can address code, variability models, requirements, test suites, etc. This should widen the vision including all the interesting works in the field.

The purpose of simultaneously requiring the two first criteria is to select papers that explicitly (and not tangentially) address methods or techniques for SPL oriented evolution. The presence/absence of these methods and techniques is evaluated for each candidate paper through reading it thoughtfully. Mere software evolution (refactoring or reengineering) is not considered if the product line context is missing. Contributions in the SPL domain which do not discuss software evolution are not considered (building SPLs from scratch, for instance). Consequently, the exclusion criteria refer to publications where any of the following items is not fulfilled:

- The publication is not focused on software evolution.
- The publication is not focused on SPLs.

Finally, relevance criteria will be applied to filter the primary sources. Only peer reviewed publications (written in English) will be considered, including journals and international conferences or workshops in the software engineering domain. Non-peer reviewed publications (including tool demonstrations, tutorials, talks, patent applications, posters, panel summaries, or keynotes) or those written in languages other than English will be discarded. Additionally, we use accepted criteria to assess the quality of the studies: they must have at least a minimal description of the context, clear objectives connected with the content of the article, and a consistent description of the research method applied. In contrast, opinion or speculative papers, Ph.D. thesis proposals (sometimes published as peer-review papers) are not considered. No time scope was considered (except the closing date of December, 2011), as one of the interests of the mapping study is to have a temporal view of the type and maturity of the research proposal.

3.4. Classification criteria

The critical aspect is to categorize the paper in a concrete approach as the base for answering the RQ1.1 research question. Obtaining a unique classification for each is very restrictive. We therefore considered a multifaceted classification, considering several aspects. We have compiled a set of non-exclusive keywords used as tags associated to each selected paper. The facets are: main focus, proposed techniques, development paradigm, and validation evidence. A set of keywords related to each of these facets is determined. Each paper is manually assigned to at most one keyword (tag) per facet except for the validation aspect.

The main focus facet can be the reengineering of legacy systems to reconfigure them as an SPL or the refactoring of an existing SPL to improve its quality. Another facet refers to the techniques proposed in the paper. The keywords used for classifying this facet are: process (from mere guidelines to tool supported processes), textual or model versions of requirements, or implementation and design techniques. If the focus is on implementation details, the language paradigm can be a significant aspect. These three aspects provide information to answer the RQ1.1 question.

On the other hand, we need to detect the degree of validation of the advances in the field (RQ1.2 research question). The validation can vary from academic examples to industrial success stories, and can include empirical evidence (see last row of Table 3). Related with these facets, Peterson et al. [11] recommend characterizing the type of research following a

Table 3
Selected keywords used to categorize the selected papers in the field of study.

Facet	Keyword
Main focus	Legacy systems reengineering into SPLs or SPL refactoring
Techniques(s) proposed	Process, requirements/models (including feature models), code/design
Development paradigm	Procedural (C), Object-oriented (C++, C#, Java), Aspect oriented (AspectJ), Feature-oriented (Ahead)
Validation evidences	Tool support, case study, industrial experience, empirical data

classification scheme, presented in [29], which uses six categories (reorganized for convenience):

1. Solution proposal, which discusses new or revised techniques.
2. Validation research, that concerns evaluating novel techniques not yet deployed in industry (an academic case study can be an example).
3. Evaluation research, which concerns evaluating a previously proposed practice (empirically or formally, better if the evaluation is conducted on industrial case studies).
4. Experience papers, which discuss how someone did something in practice, and the lesson learnt (interesting if relevant for other practitioners).
5. Philosophical papers that structure the field in new ways such as taxonomies.
6. Opinion papers.

The rationale is based on the engineering cycle of activities: problem investigation, solution design and validation, solution implementation, use and evaluation. For our purposes, the first three types are the most useful and indirectly indicate the maturity of a specific technique, process or tool, thus helping to answer the RQ1.2 question ("What is the maturity level of the approach?"). The presence of case studies or prototype tools in an academic context indicates at least a certain degree of validation. The recurrence of case studies in an industrial context or large SPLs, together with empirical evidence, demonstrate a more mature status of the method or technique (an evaluation type paper). The facet named "validation evidences" is used to classify the papers according to the types proposed above. These types constitute the keywords associated to this facet. The implicit assumption is that the lower the maturity level in a field is, the more research opportunities it offers. Thus, we can cross the information of the RQ1.1 and RQ1.2 questions to discover the research opportunities in the field.

Data tables related to the research questions are built, examining the population of the papers in relation with the tags (the map of the field). These tables synthesize results to get a set of related papers that can be analyzed in detail in a further step. During the extraction process, the tags of each primary study will be independently assigned by two reviewers. If disagreement occurs, discussions will be held among the authors until consensus is reached.

4. Search conduction and classification: Overview of the results

This section presents the details of the search conduction and classification of the papers, compiling some statistics on the primary studies. Steps 2 to 4 of the systematic mapping study are detailed: Conduction (search for primary studies, using the established criteria), Screening of the papers, and Classification of the papers, assigning diverse tags, following multifaceted criteria.

4.1. Procedure and selection

The selection of studies was performed through a multi-step process (Fig. 1):

- Searching databases to identify studies by using the search terms.
- Exclusion of irrelevant studies based on analysis of their titles and abstracts (and the full text if needed) following the topic exclusion criteria.
- Incorporation of cited references (not previously found) by reading the related work section.
- Exclusion of non-peer reviewed studies (filtering by relevancy exclusion criteria).
- Categorization and classification of primary studies by reading the full text.

The search by primary studies was conducted according to the previously established plan, using the search string in the main databases. This was automatically done, since these sources provide an efficient search engine. To support the organization of the primary studies, we used Mendeley[1] as the source reference manager system. It enables information on the primary studies to be stored on the local desktop and shared on-line (for instance, title, authors and abstract, and tags can be added to each paper).

[1] http://www.mendeley.com/.

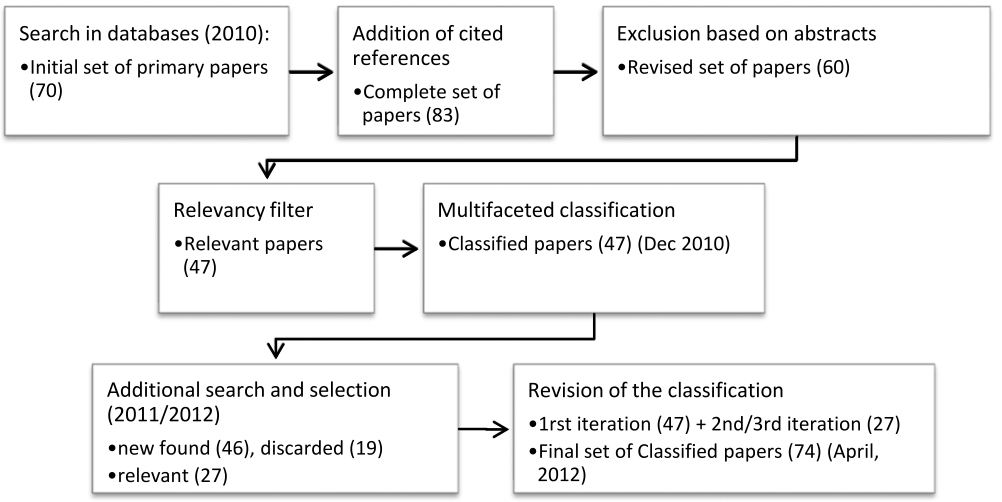

Fig. 1. Search steps for publications on *SPL oriented evolution* related papers.

Initially, during November/December of 2010, we conducted the search using the Science Direct/Scopus from Elsevier and Web of Science databases. These databases include all the reference journals in the software engineering domain. IEEE Xplore, ACM Digital Library, and SpringerLink databases guarantee the presence of the main conferences in the same domain as they specifically publish the proceedings of the most important SPL related conferences: the Software Product Line Conference (SPLC, published by Springer LNCS, ACM or IEEE in different years), the Conference on Software Maintenance and Reengineering (CSMR, IEEE), or GPCE (ACM or Springer LNCS). In the first step, we identified primary studies in these databases, following the systematic mapping study plan and the search string established previously. A total of 56 studies were found. A secondary search was conducted, using CiteseerX and Google scholar. As a result, a total of 70 studies were identified, all written in English. We adopted a flexible strategy during these initial steps (before the abstract/references analysis phase): technical reports published as similar to referred papers are readily discarded, while the ones not published elsewhere were considered during the initial steps, at least to check the references of the related work section.

The second step consisted of reading the abstract and related work sections of this initial set of publications, extending the set with cited publications (13 added, 83 in total), following a complementary reference based search [30]. At the same time, we applied the topic related inclusion/exclusion criteria to reduce the set of papers (23 were excluded) and a total of 60 studies were compiled. Finally, we selected the primary studies, filtering the set according to the relevance exclusion criteria. In this step, only peer-reviewed journal or conference papers were selected and 13 papers were excluded (theses, technical reports, presentations, tool demos, etc.). At the end of the process, 47 studies were considered as the most relevant to our systematic mapping study. In a last phase, these papers were read in full, analyzed and categorized, adding the applicable tags from Section 3.4.

At the end of 2011, and during April 2012, the search and analysis was repeated, discovering 46 new papers, mostly from 2010 and 2011. Of these, 12 were excluded based on topic criteria and 7 were discarded based on the relevance criteria (27 finally selected).

The final set of 74 studies is listed in Appendix (the excluded papers are listed as an appendix available as a technical report from the GIRO Web pages[2]). We believe that the search for publications is sufficiently extensive and that the set gives a picture of the state of the art in the field. A more detailed analysis and categorization was conducted (see Sections 5 and 6) on the 74 primary studies included in our systematic mapping.

4.2. Overview of the results

The finalization of the search and selection procedure allows us to have a global picture of the diversity of sources in the investigated field. The number of journal papers is 8, published in Information and Software Technology (4), Journal of Universal Computer Science (2), Software Practice and Experience (1), and Journal of Software Maintenance and Evolution (1). This is a number significantly lower than the number of conference publications (66), but it is clearly increasing as the field matures (Table 4 and Fig. 2). The preferred conferences are, as expected, SPLC (12 papers from the 66, from 2002 to 2011) and CMSR (6 papers, from 2004 to 2011). ICSE and GPCE are also represented. Several specific workshops on SPLs with a different focus or generic software reuse complete the landscape, having minor incidence.

2 http://giro.infor.uva.es/Publications/.

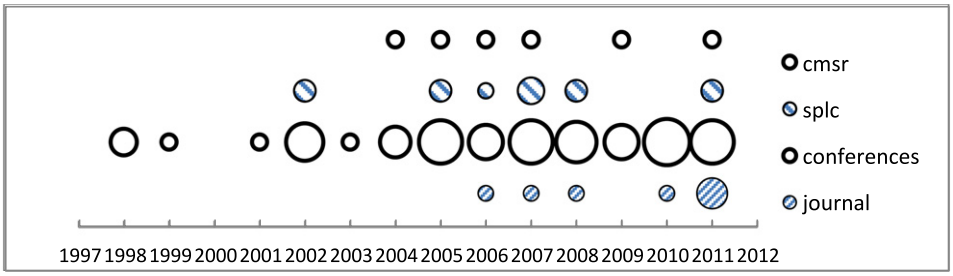

Fig. 2. Map of the temporal distribution of the sources of the selected papers [29].

Table 4
Temporal distribution of the sources (and more frequent conferences data).

Year	Journals	Conferences (all)	SPLC	CMSR
1998		3		
1999		1		
2000				
2001		1		
2002		6	2	
2003		1		
2004		4		1
2005		8	2	1
2006	1	5	1	1
2007	1	8	3	1
2008	1	7	2	
2009		5		1
2010	1	9		
2011	4	8	2	1
Total	8	66	12	6

Citation data are a direct indication of the impact of the primary studies. Table 5 provides an overview of the most cited studies, obtained from Google Scholar and Scopus. The data presented gives a rough indication of the relevance of the studies, although they cannot be considered an absolute value of quality. Only 15 studies have been cited by 10 or more. However, 22 studies were published in 2010 and 2011, so it is not expected that they can reach a higher citation number in such a short period.

The works of pioneers from the IESE institute (Bayer, Kolb, Pinzger, John) appear among the most cited, while the presence of SEI (Smith) is testimonial. It is surprising and possibly due to the SEI's interest in influencing Industry more than Academia. Also representative is the presence of some companies (see last column). However the general impression is that there are few broadly accepted methods or techniques in the field.

4.3. Classification of the approaches

One of the most important aspects of a systematic mapping is to classify the diverse approaches to get the big picture (and hence to answer the research questions). After a process of categorizing the papers, a clearer picture can appear. We have used the keywords assigned manually to the paper and the multifaceted classifications described in Section 3. Classifications of the primary studies were carried out by the first author and validated by the second. Disagreements were resolved through discussions or (during the first stages) led to refinements of the classification scheme, which in turn led to a reclassification and revalidation of previously classified publications. This procedure was repeated until no disagreements remained.

Firstly, the main orientation of the paper has been assigned to the reengineering of legacy software or existing product line refactoring: is the paper oriented towards the refactoring of existing SPLs (product line refactoring) or to the application of reverse engineering, reengineering, or restructuring of a legacy system to build the SPL? (cf. research question RQ1). In the second case, we cannot speak strictly of refactoring, as it probably implies functionality changes, or at least an evolution from hidden features to explicit mandatory/optional/alternative ones. The reengineering of legacy systems is a recurring approach, while product line refactoring is a field with, up till now, a lighter weight.

Second, we annotated the technique(s) promoted in each paper: process, requirements/models, or code/design (research question RQ1.1). Fig. 3 shows how the reengineering of legacy systems has been studied more than product line refactoring, as expected, while code and requirements have received similar interest. Finally, if the implementation is the selected level, we are interested in the language paradigm used: aspect-oriented programming (AOP), object-oriented programming (OOP),

Table 5
Cites by article (obtained from Google Scholar and Scopus) and origin of the studies (Dec., 2011).

Paper Id.	Google	Scopus	Authors	Year	Source	Research Center	Company
S61	54	15	Trujillo et al.	2006	GPCE	Austin, USA	
S35	30	14	R Kolb et al.	2005	ICSM	IESE	Ricoh
S7	27		Bayer et al.	1999	ESEC/FSE	IESE	
S4		23	Alves et al.	2006	GPCE	UFPE-Brazil	
S66	23		Zhang et al.	2003	IWPSE	Singapore	
S31	23	3	Kang et al.	2005	SPLC	Korea	
S56	16		Smith et al.	2002	SPLC	SEI	
S15	16		Critchlow et al.	2003	REFACE	Caroline, USA	
S68	15	6	Gruler et al.	2008	SPLC	TU München	
S10	13		Calheiros et al.	2007	Refactoring Tools	UFPE-Brasil	
S27	13	1	Hubaux et al.	2008	SPLC	Namur	
S70	13		Pinzger et al.	2004	PFE	IESE	Nokia
S36	12	6	Kolb et al.	2006	J. Softw. Maint.	IESE	Ricoh
S29	10		I John	2002	PFE	IESE	
S67	10	4	Breivold et al.	2008	Euromicro	Sweden	ABB
S53	8	3	Savolainen et al.	2007	ICSAC		Nokia
S32	6	2	Kim et al.	2007	SPLC		Samsung
S28	5		I John	2006	SPLC	IESE	
S20	5		Ghanam et al.	2010	Agile Processes	Alberta, Canada	
S43	4		Loesch et al.	2007	CSMR		Bosch

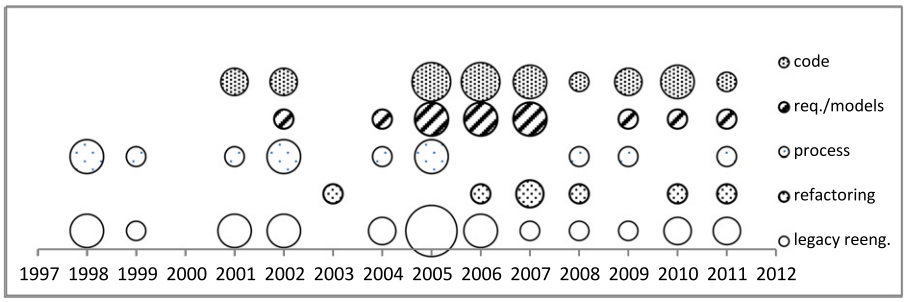

Fig. 3. Temporal distribution of legacy reenginering and product line refactoring studies and process/requirements/code comparisons.

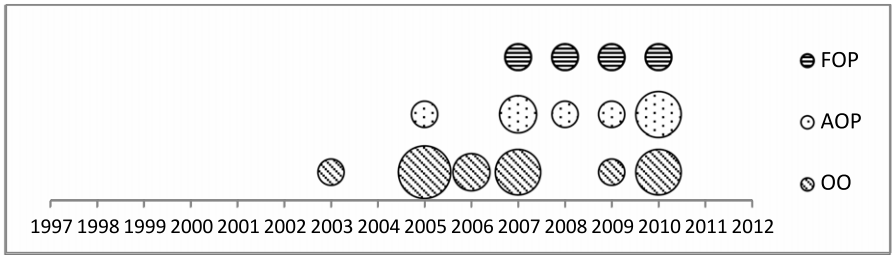

Fig. 4. Temporal distribution of OOP, AOP and FOP languages.

and feature-oriented programming (FOP) (Fig. 4). The dominant OOP paradigm is being challenged by the newer AOP/FOP approaches.

The last classification aims to evaluate the degree of maturity of the proposal, from the moment the initial idea is presented until a formalized process or technique is validated in an industrial context (including tool support and/or empirical evidence) (research question RQ1.2). We have used several indications as evidence of the type of paper:

- Does the study propose a manual process (guidelines) or does it provide detailed techniques? Is the technique formalized? Is there tool support? Are only toy examples presented? The presence of convincing case studies or mathematical proofs is required to classify the paper as a *validation* type. Otherwise it is classified simply as a *proposal*.
- If validation is present, what is the degree of validation of that proposal: Case study or industrial evidence? How large is the SPL involved? Does the study collect empirical/experimental data? Only recently have we been able to speak of empirical evidence in some papers. Both industrial experience and empirical evidence are required to include the paper in the category of *evaluation*. These papers require a deeper study to evaluate the quantitative data presented and get a

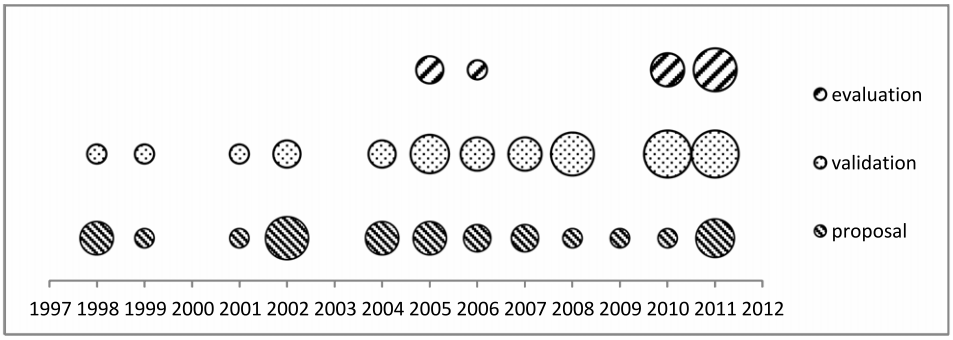

Fig. 5. Map of the temporal distribution of the validation degree of the approaches, following [29].

closer picture of them, as they suppose a qualitative advance over the apparently well founded, but on many occasions, unchecked methods or techniques.

Fig. 5 shows the temporal evolution of these characterizations without consideration of their focus or development paradigm.

Based on this classification, we analyze the most interesting aspects of the papers in detail in Sections 5 and 6 to address the research questions. The general view aims to check the status of both subfields (legacy reengineering and product line refactoring), considering the approach maturity, with special references to tool availability and use of quantitative data. In these sections, the most representative approaches are commented in detail. We consider several challenges: the process of migrating legacy code into an SPL and/or refactoring it; the techniques themselves, including model and OOP/FOP/AOP code oriented refactoring and reengineering techniques (addressing research question RQ2). To assess the techniques, although most papers include at least one case study, we specifically consider the publications that contain:

- Industrial experiences as evidence of their applicability in real situations.
- The availability of tools to support the automated reengineering/refactoring of code and/or models.
- The use of quantitative data for empirically evaluating the quality of the resulting SPL architecture.

In each section, a set of tables summarizes the articles that fulfill the classification aspects. We have prioritized the reengineering/refactoring dichotomy against model/code or other considerations, although some approaches consider the full cycle of SPL building and improvement in their process. Then, we further divide the set of papers of each section into several subgroups, using the classification criteria (the main focused technique and, in the case of implementation, the development paradigm). A discussion of the current trends and pending challenges (research question RQ2) is provided in each subsection and summarized in Section 7.

4.4. 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 [31]. 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 reengineering or refactoring, we consider that this is a controversial term and several authors use different names. We used an initial set of papers during the planning phase (starting from publications of the well-known SEI and IESE institutes) to determine the different denominations to cover as many representative variants as possible, checking the original definitions. Another aspect of the construct validity is the assurance that we actually 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 long list of different publication media indicates that the width of the search is large enough.

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, Scopus, ISI Web of Science, ACM and IEEE digital libraries, and SpringerLink specially to extract information of LNCS series, used by several important conferences as the diffusion medium). The inclusion/exclusion criteria are related to whether the topic of the field is present in the paper or not. The classification is another source of threat to reliability, but the consistency of the classification is ensured by having the classifications conducted by a first author and validated by a second. Internal validity is concerned with the analysis of the data. Since the analysis only uses descriptive statistics, the threats are minimal.

Table 6
Studies on process for reengineering legacy systems into SPLs.

Ref.	First Author (year)	Method and/or SPL case study	Inst.	Validation
[S17]	DeBaud (1998)	Promotes Reengineering as entry point to SPL	SEI	
[S62]	Weiderman (1998)	Guides for organizations	SEI	
[S73]	Scherlis (1998)	Design management and manual techniques	SEI	
[S57]	Stoermer (2001)	MAP method	SEI	case study
[S46]	O'Brien (2002)	OAR method	SEI	
[S56]	Smith (2002)	OAR method case study	SEI	case study
[S7]	Bayer (1999)	RE-PLACE method	IESE	industry
[S71]	Raghavan (2002)	Common interfaces separated from components	IESE	
[S24]	Hansen (2004)	Workshops for reengineering		case study
[S70]	Pinzger (2004)	Process based on PULSE method.	IESE	industry
[S34]	Knodel (2005)	Asset Recovery	IESE	
[S33]	Knodel (2005)	Analyzing Asset Adequacy	IESE	industry
[S67]	Breivold (2008)	Based on dependency analysis. Evaluates quality attributes (Analyzability, Extensibility, etc.), ABB		industry
[S31]	Kang (2005)	Home Service Robot, HSR SPL		case study
[S40]	Lee (2009)	Set-top box manufacturing company SPL		industry
[S30]	Jun (2010)	Court Business Management System SPL		industry
[S26]	Hubaux (2008)	Assessing the variability mechanism in PloneMeeting SPL case study		case study
[S27]	Hubaux (2008)	Separation of concerns, PloneMeeting SPL		case study
[S20]	Ghanam (2010)	Agile process using tests, Buddi SPL		case study
[S9]	Bosch (2011)	Agile process, Intuit's Quickbooks SPL		industry
[S6]	Bartholdt (2011)	Reengineering of a hierarchical Siemens healthcare imaging SPL		industry
[S63]	Wu (2011)	Clone detection process, DirectBank SPL		industry
[S74]	Zhang (2011)	Incremental and agile process, Alcatel-Lucent SPL		industry

5. Legacy system reengineering into SPLs: Classification and discussion of the approaches

5.1. Methodological work in the reengineering process

Table 6 includes the articles devoted to the description of methods for reengineering legacy applications into SPLs and the experiences that use them. They use conventional reengineering techniques or adapted ones, but the study is predominantly focused on the global process. The table contains the pioneering works from the SEI and IESE Institutes. They are included in this mapping study due to their historical importance in building a reuse infrastructure and an SPL culture.

The proposals related to the SEI include the papers of De Baud et al. [S17], Weiderman et al. [S62], and Scherlis [S73] that proposed generic guides for reengineering processes. More concrete approaches were the MAP (*Mining Architectures for Product lines*) method [S57] and the OAR (*Options Analysis for Reengineering*) method [S46]. Case studies were also presented following OAR [S56] and MAP using the Dali workbench [32] to extract and reconstruct architectures.

The proposal from the IESE, RE-PLACE (Reengineering-Enabled Product Line Architecture Creation and Evolution) [S7] is an approach to support the transition towards SPL architecture. The authors illustrate the approach with an industrial experience, using commercial reengineering tools. Knodel et al. [S34] [S33] integrate existing documentation and assets into an SPL. In [S33], they specifically propose the use of ADORE (Architecture and Domain-Oriented Reengineering), a request-driven reengineering approach.

The rest of the papers in this group are basically case studies or industry experiences that present lessons learned while applying those guides. Raghavan et al. [S71] (Nokia), Pinzger et al. [S70] (Nokia/Philips in collaboration with IESE), and Breivold et al. [S67] (ABB) show some of these experiences. Kang et al. [S31], Jun et al. [S30] and Lee et al. [S40] describe successful industrial reengineering projects that use conventional reengineering methods and CASE tools. Hansen et al. [S24] focused on organizational challenges, related to the large number of heterogeneous stakeholders involved in the reengineering effort.

However, in the last few years, new alternatives have been proposed, promoting agile software development (Bosch et al. [S9], Ghanam et al. [S20]), separation of concerns (Hubaux et al. [S26] [S27]), hierarchical SPLs (Bartholdt et al. [S6]), or a bottom-up process based on clone detection (Wu et al. [S63]). Zhang et al. [S74] uses an incremental and agile process in an Alcatel–Lucent SPL reengineering project. In this case, quantitative data are collected before and after reengineering and return of investment is estimated in more than 100%. These proposals indicate that there is a renewed interest in process adaptation to AOP techniques or agile processes, traditionally away from systematic development. The influence of these works will soon be seen but we think that, in general, the available data is neither conclusive nor detailed enough, so it remains an open issue.

When available, quality improvements are limited to the presented case studies. However, the company global improvement due to the adoption of a systematic method to migrate legacy systems to SPLs is not a proven fact. Documenting the increase in productivity and/or quality requires the company to have previous measurements of these data (level 4 in the CMM SEI model), something that is regrettably not very common in industry. It would be desirable

Table 7
Studies on the extraction of SPL models from legacy documentation or code.

Ref.	First author (year)	Focus	Tool support	Validation
[S29]	John (2002)	Legacy Documentation		industry
[S47]	Pashov (2004)	Feature models guide architecture recovery, Image processing application		industry
[S34]	Knodel (2005)	Asset Recovery and Incorporation into Product Lines		
[S1]	Ajila (2005)	Feature extraction		
[S28]	John (2006)	Legacy Documentation		industry
[S32]	Kim (2007)	Feature models guide architecture recovery, Digital Audio and Video SPL		industry
[S13]	Conejero (2009)	Analysis of crosscutting		
[S58]	Stoiber (2010)	Feature Unweaving, automated model refactoring	ADORA [33]	
[S11]	Chen (2005)	Feature analysis techniques using reengineering tools	RECON [34] FEAT [35]	case study
[S23]	Graaf (2006)	Migration by Model Transformations	ATL [36]	case study
[S39]	Kästner (2007)	Feature-oriented analysis of legacy code	CIDE	

for companies with practices involving both fields to share their (positive or negative) experiences with the software engineering community.

5.2. Model oriented reengineering and extraction

Table 7 summarizes the articles devoted to the reengineering of existing models and the required documentation to establish a new SPL. We consider two main subgroups of papers: (a) papers that use legacy documentation to build the SPL models and documentation, mainly in a manual manner, and (b) papers that use legacy code and tools to extract SPL models and documentation.

Concerning the first group, the works of John [S29], [S28], and Knodel et al. [S34], try to integrate existing documentation into an SPL. In practical terms, a catalog of existing techniques taken from the reengineering world is adopted. The approaches of Pashov et al. [S47] and Kim et al. [S32] share the idea of imposing feature models to guide the migration towards a component-based architecture of legacy systems. However, they use feature models to provide an orientation for the establishment of architectural hypotheses. The papers by Ajila [S1] and Conejero et al. [S13] deal with the extraction of features from existing products, using boundary classes [S1] or traceability matrices [S13] to detect crosscutting. The process remains mainly manual. However, in Stoiber et al. [S58], *feature unweaving* is used to extract variable features using a graphical tool based on the ADORA language [37]. Their tool [33] can automatically refactor an ADORA model into one in which the elements of each feature are grouped into an aspect. The advantage of having only one requirement model is the counterpart of its evident complexity, although it is an interesting alternative to mainstream approaches based on complementary (feature based) variability models.

The extraction of SPL models from a legacy system can benefit from the experience in reengineering work. In Chen et al. [S11], dynamic and static feature analysis is used to locate features in implementation modules. These techniques can be supported by commercial and free tools, such as *RECON* (Software Reconnaissance) [34] or *FEAT* (Feature Exploration and Analysis tool) [35]. Graaf et al. [S23] migrate legacy architectures towards an SPL, applying an automatic model transformation. They define transformation rules written in *ATL* (Atlas Transformation Language) [36]. The migration is only feasible if the variability can be defined by a meta-model. Finally, Kästner et al. [S39] propose a tool, *Colored IDE* (now known as *CIDE*) to automate the extraction of features from legacy applications, once the fragments are marked. However, the process remains manual and depends on the experience of the user. The interest of the tool probably lies more on feature-oriented refactoring (see Section 5.3).

With these experiences, we can conclude that reengineering techniques can be used advantageously to extract features. Tools such as *RECON* or *FEAT* can detect reengineering opportunities. An integration of these techniques in a guided process of feature model extraction (instead of plain features) is a reachable challenge that justifies an effort of systematization. The *CIDE* tool would benefit from this integration, if the code could be automatically linked to a feature. This is commented on in more detail in the following sections. Table 8 summarizes the mentioned tools (Dali is a tool built specifically to support the SEI MAP method) and Fig. 6 shows a global picture (*Concept Explorer* is used in product line refactoring and is introduced in Section 6.2).

5.3. Work on legacy code reengineering

Table 9 includes the articles devoted to the reengineering of legacy code into Object-oriented (OOP), Feature-oriented (FOP), or Aspect Oriented (AOP) program artifacts that will constitute the initial architecture of the SPL. Classic OOP is clearly a more mature field compared with the alternatives, but recent trends indicate a greater presence of the AOP paradigm. FOP can be seen as an improvement of OOP (adapting common languages like Java), and less radical than AOP. Empirical studies are also abundant, facilitating the assessment of the approaches. For a better organization, we have divided the analysis into generic OOP (including FOP) and AOP reengineering papers (Sections 5.3.1 and 5.3.2 respectively), articles that define metrics (sometimes with associated refactorings, Section 5.3.3), devoting Section 5.3.4 to summarizing the state of the art, the available tools and the pending challenges.

Table 8
Summary of tools used for SPL model extraction.

Ref.	Tool	Main utility	Availability	Observations
[36]	Atlas Transformation Language (ATL)	Model transformation	Open source	MDE based conversion of models
[37]	ADORA extension	Model refactoring	Open source	Nonstandard requirement tool
[34]	RECON	Reengineering, Feature detection	Free	Used to detect features in code
[35]	FEAT	Concern graphs	Free Eclipse plug-in	To find structural program dependencies.
[S39]	CIDE	Code refactoring	Free Eclipse plug-in	Features are identified with different colors
[32]	Dali	Reengineering,	SEI integrated tool	Architecture extraction, support of MAP method

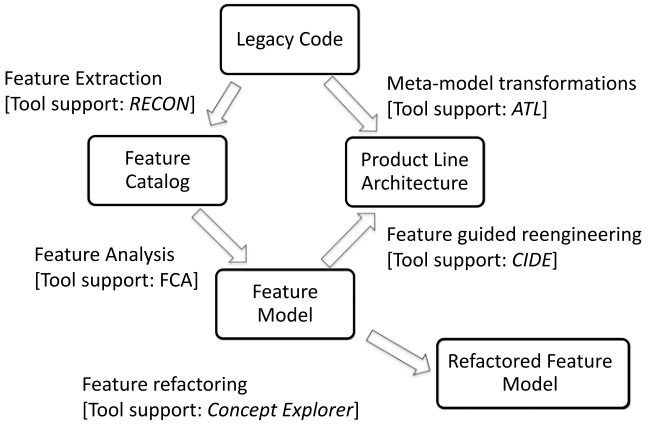

Fig. 6. Models involved on the reengineering of legacy systems into SPLs and examples of supporting tools.

5.3.1. OOP (and FOP) reengineering of legacy code

Zhang et al. [S66], Simon et al. [S55], Akers et al. [S2], and Olszak et al. [S45] propose different tools to analyze and migrate legacy systems into an SPL, profiting from reengineering techniques and tools and using conventional OO languages. In [S66] they use their tool, XVCL, to build a mobile SPL, looking for portability across J2SE and J2ME platforms. Simon et al. [S55] use a tool for feature analysis (XFIG) and a tailored reengineering technique, based on Formal Concept Analysis (FCA). Akers et al. [S2] describe the Design Maintenance System (DMS), a commercial tool, and discuss how it was applied to a large C++ avionics system, transformed by Boeing into an SPL of embedded systems. DMS is a complete reengineering tool that performs analyses and manipulations on the abstract syntax tree representations of programs. Chiang et al. [S12] develop a distributed object computing architecture for an industry SPL by reusing legacy data-centered systems through wrapping techniques. Olszak et al. [S45] present a method and two real-world case studies for feature-oriented re-modularization of Java programs. They use execution traces to locate features, and Java packages to establish explicit feature modules. However, no complete separation of features is achieved.

Some papers claim to provide refactoring techniques to build an SPL, although, due to the difficulty of comparing the behavior of a (family of) monolithic product(s) and a complete SPL, we consider them as reengineering.

Several articles propose to use the FOP paradigm [38] to manage features at the code level, using a modified version of Java and the Ahead Tool Suite (ATS). Liu et al. [S41] define a theory of feature-oriented refactoring (FOR) as the process of decomposing a program into features (the inverse of feature composition). The practical process organizes the code by assigning every class member to a feature under the guidance of an Eclipse based refactoring tool. However, this tool is not publicly available. Trujillo et al. [S61] present a case study in manual feature refactoring of the ATS itself. Each feature encapsulates a tool of the suite and includes sources, makes files, HTML documentation, and regression tests specific for that feature. Lopez-Herrejon et al. [S44] have proposed a catalog of refactorings to extract variable features from existing single systems. The catalog includes refactorings such as *Addition at the beginning, Addition at the end of the method, Addition anywhere with a hook method, Overwrite method*, etc. (see Table 10). The goal is to reorganize the code to be used in the context of FOP (Feature IDE [39] is used to compose the variants).

More specific approaches introduce new possibilities. Ghanam et al. [S20] introduce an agile approach to reactively refactoring the existing code using tests to discover the variability. They provide an Eclipse plug-in (*ProductLineDesigner*). The limits of the current status of the plug-in are the inability to combine variation points in a hierarchical manner or to support dependencies between variation points and variants. Dabholkar et al. [S16] present the Feature-Oriented Reverse Engineering for Middleware Specialization (FORMS) approach. It is based on source code analysis and allows domain concerns to be modularized in three stages: Feature mapping, Discovering Closure Sets, and Middleware Composition

Table 9
Summary of legacy code reengineering studies.

Ref.	First author (year)	Focus (language)	Tool	Validation
Generic OOP/FOP/AOP				
[S66]	Zhang (2002)	Reengineering into a mobile device SPL (J2SE to J2ME)	XVCL	case study
[S55]	Simon (2002)	Analysis of the legacy code using FCA	XFIG	
[S12]	Chiang (2004)	Developing a distributed architecture through wrapping techniques		industry
[S2]	Akers (2007)	Reengineering tools are used to build and Avionic SPL (capturing and manipulating C++ code structure)	DMS	industry
[S45]	Olszak (2009)	Finding features, following execution traces to locate them and packages to separate feature modules (Java) BlueJ and JHotDraw examples	-	case study
[S41]	Liu (2006)	Feature-oriented refactoring theory and application in Prevayler SPL (Java and Ahead)	FOR (?)	formal, case study
[S61]	Trujillo (2006)	Refactoring code and documentation into a product line (Ahead)	-	case study
[S44]	Lopez-Herrejon (2011)	Refactoring to extract SPLs using FOP (Java and Ahead)		case study
[S20]	Ghanam (2010)	Test-Driven Approach for refactoring (Java)	Eclipse plugin	case study
[S16]	Dabholkar (2010)	FORMS method for middleware framework (cross platforms)	MPC	case study
[S65]	Xue (2011)	Variability analysis to detect clone and feature location (Java)		
[S37]	Kästner(2007)	refactoring of the Berkeley DB into 38 features (AspectJ)		case study,
[S39]	Kästner (2007)	Feature-oriented refactoring (AspectJ)	CIDE	
[S38]	Kästner (2009)	Virtual and physical separation of concern equivalence (Light Java)		formal, case study
[S5]	Alves (2005)	Extracting and Evolving Mobile Games Product Lines (AspectJ)	FLiP	case study
[S10]	Calheiros (2007)	Product line variability refactoring tool (AspectJ)	FLiP	case study, tool
[S3]	Alves (2008)	Extraction of SLP with Aspects (AspectJ)	FLiP	tool
[S59]	Tizzei, (2010)	An Aspect-oriented View to Support Feature-oriented Reengineering		case study
[S69]	Lozano (2011)	Survey of the approaches to detect variability in source code		
Metrics based guidance and empirical evidence				
[S19]	Ganesan (2005)	Taxonomy of metrics to analyze legacy code (none)		
[S35]	Kolb (2005)	Reengineering legacy code (C++)		case study
[S36]	Kolb (2006)	Reengineering legacy code (C++)		case study
[S8]	Berger (2010)	Measuring the feasibility of existing code to extract an SPL (none)		
[S14]	Couto (2011)	Extracting an SPL from Argo UML system using conditional compilation (Java) and standard metrics	Compiler Standard metrics	case study empirical
[S51]	Saraiva (2010)	Feature models and metrics guide the transformation process into Ginga SPL (AOP platforms)		industry

Synthesis, using an advanced cross-compiler such as MPC (Make Project Creator). Xue [S65] proposes to use existing tools for clone detection and feature location to automate most of the manual work in the analysis of existing products. Finally, Lozano [S69] has recently surveyed the approaches to detect variability concepts in source code, organized around variability concepts (features and variants).

5.3.2. Aspect oriented reengineering of legacy code

In the last few years, an increasing trend is the use of aspect oriented programming (AOP) techniques to implement the variability of an SPL.

We have already cited the *CIDE* tool of Kästner et al. [S39], used to mark feature code in legacy Java systems. For the implementation, they use AspectJ oriented refactoring as *Extract Introduction* or *Extract Advice*. In a related paper [S37], Kästner et al. evaluate AspectJ using a case study that refactors the embedded database system Berkeley DB into 38 features. They used several OO to AOP refactorings collected in Table 10 (*Extract Introduction* refactoring to move methods or fields to aspects and *Extract Beginning*/*Extract End* refactorings to create advice and extend join points throughout the code are the most common). However, the final result of the experience was negative for the authors, due to problems with the AspectJ language itself, and the authors advocate alternative solutions, closer to conventional OO approaches: In a posterior work [S38], they provide a model that supports both physical (Ahead based) and virtual (CIDE) separation of concerns and formally define refactorings in both directions to show the equivalence.

Alves et al. [S5][S3] and Calheiros et al. [S10] extract a product line, refactoring a case study in the domain of mobile games, where variations are handled with aspect oriented constructs. They build a variability concern graph using the mentioned *FEAT* tool [35]. The discovered concerns guide the refactorings proposed by Alves et al. (*Extract Method/Resource to Aspect*, *Extract Context*, *Extract Before/After Block*, etc). In [S3] and [S10], the authors complement their work with *FLiP*, a tool suite implemented in Eclipse. One of the components, *FLiPEx*, is the refactoring tool that allows variability to be extracted from Java classes into aspects using a wizard. *FLiP* connects with a commercial SPL tool (Pure::Variants) to manage the features.

Table 10
Some refactoring examples proposed by Alves et al. [S5], Kästner et al. [S39], [S37], Lopez-Herrejon [S44], Kolb et al. [S35][S36].

Intention	Ref.	Refactorings
Aspect oriented refactoring	[S5]	Extract Method/Resource to Aspect, Extract Context, Extract Before/After Block, Move Field to Aspect, Move Import Declaration to Aspect, Move Interface Declaration to Aspect, Move Method to Aspect, Move Extends Declaration to Aspect
Aspect oriented refactoring	[S39] [S37]	Extract Introduction, Extract Advice, Extract Beginning, Extract End, Extract Before/After Call
Feature extraction (FOP)	[S44]	Addition at the beginning, Addition at the end of the method, Addition anywhere with a hook method, Overwrite method , Move entire method, Move field, Remove field modifiers declarations, Move entire class
Improve general quality	[S35] [S36]	Renaming of files and functions, splitting of long files, moving of functions from one module to another, conversion of macros to inline functions, changing of data type
Improve maintainability and reusability	[S35] [S36]	Removal of internal and external code clones Merging of different implementations and realization using conditional compilation Reduction of the scale and complexity of functions.

AOP based design of SPLs (alone or in combination with other paradigms such as FOP) presents some advantages as against the pure OO alternatives. However, the use of AspectJ itself seems to have problems [S37], which can be solved by combining aspects with other paradigms (components and *mixin layers* are proposed). Tools, such as*FLiPEx*, can help to refactor legacy code into an SPL, using the AOP paradigm. A clear research opportunity is the clarification of the several practical approaches used to implement the features of an SPL. Virtual separation of features (as in CIDE and standard OO implementations using compiler directives or similar techniques) has shown its usefulness in some industry experiences. Physical separation of features means a clear identification of the variability. The concrete separation of variable features using AOP (AspectJ) or FOP (with Java composition as in Feature IDE or Ahead) should be assessed, although the few existing evidences seem to favor the FOP approach.

A catalog of specific AO refactorings is already available, though more complex transformations could be envisioned. A deeper evaluation between the AOP catalog and the mentioned FOP catalog [S44] is required and the unification of the refactoring definitions will be a desirable side effect. The field is alive and we expect more advances will occur in the next few years. Table 10 summarizes the proposed refactoring (including some mentioned in the next section).

5.3.3. Empirical analysis and associated refactoring

The common consideration of the papers in previous sections is that they lack empirical guidance to detect the required refactorings or experimental evidence of the improvements achieved. Some papers, however, involve both metrics and refactoring techniques, which are closely related. Table 11 includes a summary of the collected metrics.

As a paradigmatic example, Kolb et al. [S35][S36] present an industrial case study of the application of IESE's PuLSE approach in migrating legacy software components. Both papers describe in detail the analysis of existing components' C++ implementations with respect to different quality aspects: flexibility, reusability, and maintainability. A set of source code metrics and the Goal Question Metric approach [40] were chosen. A goal is explicitly expressed, questions are formulated to give an indication of goal fulfillment, and metrics are collected for each question. Some of these metrics are: *Number of functions calling a certain function (Fan-In), number of distinct functions called by a function (Fan-Out), cyclomatic complexity* (Table 11). The quality improvement was achieved by performing automatic improvements (conditional preprocessor statements) or manual refactorings (renaming of files and functions, splitting of long files, etc.). More advanced refactorings, such as removal of internal and external code clones, have been performed in a second step. Table 10 includes both groups of refactoring techniques. As a result of the analysis and refactoring activities, the component has been improved, as shown by the quantitative data collected at the end of the activities. In a related work, Ganesan et al. [S19] describe how to identify domain-specific reusable components that have quality attributes like functional usefulness, readability, testability, etc. Each one can be broken into measurable items (for example, the metrics NOC, OCAEC, CALL_IN and DIT metrics of Table 11 are chosen to measure the usefulness of an OO component). Berger et al. [S8] propose a theoretical basis to derive a metrics for evaluating the ability of similar software products to form a product line. The SoC (*Size of Commonality*) metric can be calculated by comparing the component signatures (two components are syntactically identical if they have the same signature). Starting from that basic metric, others can be derived, including *Reusability Benefit*.

Couto et al. [S14] extract an SPL from the Argo UML system using conditional compilation and they obtain metrics on the extracted SPL. The resulting features are characterized using Size, Granularity, Crosscutting, and Localization Metrics. The last two groups are included in Table 11 and can be used to assess the SPL quality. Saraiva et al. [S51] promote the use of predefined feature models as a guide to the refactoring of the middleware into an SPL in the context of an industrial experience. The identification of crosscutting concerns in code is based on the work of Conejero et al. [13] and code metrics: *Coupling between Object Classes, Messaging Passing Coupling, and Data Abstraction Coupling*. Synchronization, resource management and transmission/distribution of the information are the three concerns identified and extracted using AOP techniques.

Table 11
Summary of metrics to evaluate the usefulness and quality of reusable assets, adapted from Saraiva et al. [S13], Ganesan et al. [S19], Kolb et al. [S35][S36], Berger et al. [S8], or to characterize an SLP, Couto et al. [S14].

Metric	Ref.	Description
COC	[S13]	Coupling between Object/Classes
MPC	[S13]	Messaging Passing Coupling
DAC	[S13]	Data Abstraction Coupling
NMPUB	[S19]	The number of public methods implemented by a class.
WMC	[S19]	Cyclomatic complexity of a class.
NOC	[S19]	The number of children/grandchildren of a class.
DIT	[S19]	The level a class is located from the root in the inheritance hierarchy.
OCAEC	[S19]	The number of times a class has been used as an attribute in other classes.
CALLS_IN	[S19]	The total number of times the methods of a class was called by other classes.
LCOM	[S35][S36]	Cohesion—The number of sets of methods that access the same attributes.
LOC	[S35][S36]	The total number of lines in a module or function
LOMC	[S35][S36]	The total number of maintainable lines in a module or function (without blank lines and comment lines)
FAN-IN	[S35][S36]	The number of distinct functions calling a certain function.
FAN-OUT	[S35][S36]	The number of distinct functions called within a function.
Cumulated data	[S35][S36]	The total number of members in a structure cumulated over its Sub-structures
Associations	[S35][S36]	Associations with other structures
Max. nesting	[S35][S36]	The maximum nesting level of control constructs
SoC	[S8]	Size of Commonality,
IoC	[S8]	Impact of Commonality
PrR	[S8]	Product-related Reusability
IPrR	[S8]	Impact of Product-related Reusability
RB	[S8]	Reusability Benefit.
SD	[S14]	Scattering Degree (Crosscutting Metrics)
TG	[S14]	Tangling Degree (Crosscutting Metrics)
StartMethod	[S14]	Localization Metrics, counts the statements annotated
EndMethod:	[S14]	Localization Metrics
BeforeReturn	[S14]	Localization Metrics
NestedStatememt	[S14]	Localization Metrics

A work in normalization and level organization of the diverse metrics is missing and, consequently, it represents an opportunity for systematizing the accumulated knowledge. On the other hand, the unified catalog would be useful to compare the SPL based on OOP classical languages with the FOP and AOP based SPLs.

5.3.4. Summary of tools for legacy code reengineering

Finally, Table 12 and Fig. 7 show the tools that have been used to extract an SPL from legacy code. Some are Aspect Oriented (lower part) and some others Object/Feature-oriented (upper part), including some SPL improving tools mentioned in the next section. Reengineering classical tools have their role as the first step (feature detection) towards an SPL. Recent tools, specific for SPL related refactoring, are appearing, most of them open source and/or Eclipse plug-ins. FlipEX is part of a suite to guide code refactoring into physical features while CIDE allows manual annotation of code to achieve virtual separation of the same features.

6. Classification and discussion on SPL refactoring studies

6.1. SPL model refactoring

Table 13 summarizes the papers that focus on refactoring models and requirement documentation to improve an existing SPL. An increase in interest on the topic is detected as industry SPLs are beginning to be operative and maintenance is required.

Alves et al. [S4] focus mainly on refactoring feature models of existing SPLs. They expand the definition of refactoring as "a change made to the structure of a software product line in order to improve its configurability, make it easier to understand and cheaper to modify without changing (though the configurability degree can be increased) the observable behavior of its original products". Several feature refactorings are described: *Convert Alternative to Or*, *Collapse Optional and Alternative to Or*, etc. Similar in their goal, Loesch et al. [S42], [S43] present a method to optimize the variability provided in an SPL, classifying the use of variable features in real products derived from a large industrial SPL. They use Formal Concept Analysis (FCA) to perform an analysis of variability, which is restructured and simplified, using refactoring strategies, such as *Turn variable features into mandatory features, Turn variable into alternative feature,* etc. In this case, model evolution guides the transformations on the source code. For variability analysis, they have adopted the Concept Explorer tool [41]; while for the implementation of the variability, they use conditional compilation standard tools. Xue et al. [S65] use a model differencing algorithm to identify evolutionary changes that occurred to features of different product variants, using a case study and empirical data. Savolainen et al. [S53] focus on analyzing SPL dependencies for identifying unnecessary features. If these

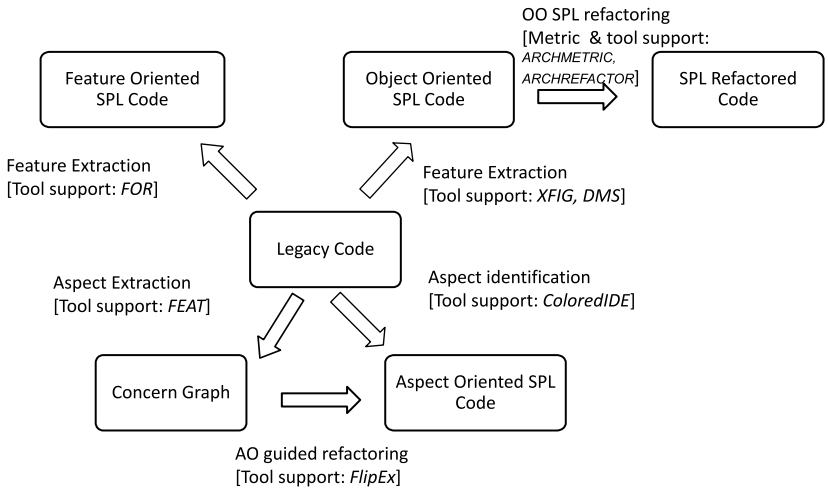

Fig. 7. Relationships between legacy code and AO, FO, OO restructured code. Examples of tools that support the different reengineering techniques are given.

Table 12
Summary of tools used in legacy code reengineering papers.

Ref.	Tool	Main utility	Availability	Observations
[S2]	DMS	Design Maintenance System (Reengineering)	Commercial	Customizable as an SPL tool
[S23]	Concept Explorer	Formal Concept Analysis	Free	Generic tool, requires preparation of data
[S39]	CIDE	Code refactoring	Open source	Features are identified with different colors
[S55]	XFIG	Reengineering, Feature detection	Open Source	Formal Concept Analysis
[S3][S10]	FlipEX	Aspect extraction	Eclipse plug-in	Wizard to transform Java code into AspectJ
[S66]	XVCL	SPL meta-composition	Open Source	Java inter-mixed with XVCL commands
[S41]	FOR	Feature extraction and composition	Not available	FOP oriented

Table 13
Summary of papers on SPL model refactoring.

Ref.	First author (year)	Paper focus and context	Tool	Validation
[S4]	Alves (2006)	Refactoring feature models, mobile games SPL	-	case study
[S42]	Loesch (2007)	Optimization of Variability in Software Product Lines based on FCA, engine control SPL	Concept Explorer	industry
[S43]	Loesch (2007)	Restructuring Variability in Software Product Lines		industry
[S53]	Savolainen (2007)	Finding and solving SPL dependencies	-	case study
[S64]	Xue (2010)	Model differencing in WFMS SPL	GenericDiff	case study empirical
[S72]	Romanovsky (2011)	Refactoring of SPL technical documentation	DocLine	case study
[S21]	Gheyi (2008)	Defines algebraic laws for feature models		formal
[S22]	Gheyi (2011)	Checking feature model refactorings	Alloy Analyzer	formal
[S48]	Peng (2011)	Evolution of variability in CAGS SPL context	-	formal
[S68]	Gruler (2011)	Extends CCS to SPL	-	formal

features are found, the method also addresses how to correct the situation. Romanovsky et al. [S72] extend the idea of SPL refactoring to technical documentation and describe their implementation with the DocLine toolset, that uses an XML-based language (Documentation Reuse Language, DRL), intended for writing reusable documentation.

Gheyi et al. work in feature model refactoring [S21][S22], which is formally grounded, encoding encode feature models into the Alloy formal specification language. The Alloy Analyzer tool, which performs analyses on Alloy models, can be used to automatically check whether encoded general and specific feature model refactorings are correct. Peng et al. [S48] propose a problem-oriented and value-based analysis method for variability evolution analysis. The method takes into account both kinds of changes (requirements and contexts) during the life of an evolving software product line. Gruler et al. [S68] extend the Calculus of Communicating Systems (CCS) [42] with variation points to formally model an SPL. Then they establish algebraic laws that can be applied to refactor SPL expressed in the extended CCS formalism. The rules allow common parts of an entire SPL to be derived. The approach is applied to a case study from the automotive domain.

Table 14
Some feature refactoring examples proposed by Alves et al. [S4], Loesch et al. [S42], [S43], and Gheyi et al. [S21][S22].

Intention	Ref.	Refactorings
Feature model Refactorings		
SPL evolution	[S4]	Convert Alternative to Or, Collapse Optional and Alternative to Or, Add New Alternative
Optimizing variability	[S42][S43]	Turn variable features into mandatory features, Remove variable features, Turn variable into alternative features, Combine variable features
Checking correctness	[S21][S22]	Any

Table 15
Summary of SPL code refactoring papers.

Ref.	First author (year)	Focus	Tool	Validation
[S18]	Frenzel (2007)	Adaptation of the Reflexion Method for improving an SPL in automotive industry	*Reflexion* tools	industry empirical
[S52]	Savga (2008)	Discussion on refactoring in SPL		
[S54]	Siegmund (2010)	Optimizing Non-functional Properties of Software Product Lines by means of Refactorings		case study
[S15]	Critchlow (2003)	Refactoring product line architectures, based on metrics guidance	ARCHMETRIC ARCHREFACTOR	case study
[S49]	Ribeiro (2008)	Tool for recommending refactorings	FliPEx	
[S50]	Ribeiro (2009)	Improving Guidance when refactoring an SPL	FliPEx	case study
[S59]	Tizzei, (2010)	Aspect oriented view of FOP		
[S60]	Tizzei, (2011)	Assessing design quality of an SPL. Components support	Conventional metrics tool	case study empirical
[S25]	Heider (2010)	Simulation of different evolution in IAS SPL	Simulation tool	empirical

The refactoring of feature models is a relatively simple task that has been exhaustively treated. The proposed formal techniques (FCA for feature model simplification, Alloy Analyzer to check the correctness, or CCS to refactor the SPL) guarantee that the refactored feature models are correct and optimal. We think that the challenge is to integrate these techniques with the code oriented tools of the next subsection to guide the more puzzling code refactoring (Table 14).

6.2. SPL code refactoring

Table 15 includes the articles devoted to the evolution of existing SPLs, using refactoring techniques, summarized in Table 16. We have again divided the analysis into generic refactoring and articles that define associated metrics. Table 17 contains some of these metrics.

Frenzel et al. [S18] adapt the *Reflexion* reengineering method [43] to reconstruct and improve the static architecture of variants in an existing SPL. The method uses clone detection and function similarity tools to map code to architecture variants. Savga et al. [S52] discuss the role of refactoring in maintaining features and their implementations. They propose integrating existing work on both levels of refactoring into a framework for uniform program transformation and tools that combine both paradigms. No concrete tools or techniques are presented however. Siegmund et al. [S54] present an approach for optimizing non-functional properties in software product lines, using refactorings. The aim is to provide differently optimized variants of an SPL. Refactorings are categorized according to their influence on non-functional properties: the *Inline Method* refactoring can increase the performance; however, it has a negative effect on binary size that can be decreased by selecting the *Pull up Method* refactoring. They have developed *Refactoring feature modules* (RFMs), a technique to define and reuse refactorings integrated with feature-oriented programming (FOP). A tool is currently being developed.

Critchlow et al. [S15] proposed a set of rather simple refactorings (*Remove optional component, Remove unused variants*, etc.). However, they use a service utilization metrics and two tools: ARCHMETRIC, which automatically calculates and presents the service utilization metrics for a particular SPL, and ARCHREFACTOR, which implements a set of pre-defined refactoring strategies to solve the problems identified by the metrics. The metrics used are PSU, *Provided Service Utilization*, which calculates the percentage of provided services of a component that are actually used by other components in a given configuration, and RSU, *Required Service Utilization*, which measures the percentage of required services of a component.

Ribeiro et al. [S49][S50] present a decision model and tool that may improve modularity and remove bad smells such as *cloned code* in SPLs. They use concern metrics to evaluate *Modularity*, *Source Code Size*, and *Scalability* criteria of the possible implementation refactorings. Inheritance, Decorator pattern, Mixins, and AOP mechanisms are considered and compared within a case study. *End of Method Body* and *Middle of Method Body* problems are focused on.

Tizzei et al. [S59] propose an *Aspect-oriented feature view*, which manages the traceability from feature models to aspect oriented SPL models. The result is a modified feature diagram where crosscutting features and their influence on other features are visualized. They also introduce, in [S60], the use of metrics to assess the stability of an SPL. They use a Component/AO mixed model, taking advantage of their characteristics in order to improve modularity. They quantitatively compare four design and implementation techniques: OO, AO, component based, and a combination of component-based

Table 16
Some code refactoring examples proposed by Savga et al. [S52], Siegmund et al. [S54], Critchlow et al. [S15].

Intention	Ref.	Refactorings
Uniform transformation	[S52]	MakeMandatory, MakeAlternative, DeleteFeature, CopyFeature, ReduceGroupCardinality.
Performance	[S54]	Inline Method, Inline Class, Remove Middleman, Remove Setting Method, Replace Delegation with Inheritance, Replace Temp with Query, Inline Temp
Styling Guidelines and Code Metrics	[S54]	Extract Method, Replace Conditional with Polymorphism, Replace nested Conditional with Guard Clauses, Extract Method, Create Template Method, Consolidate Duplicate Conditional Fragments
Footprint	[S54]	Collapse Hierarchy, Pull up Constructor Body, Pull up Field, Pull up Method, Remove Middleman, Remove Setting Method
Readability	[S54]	Extract Class, Extract Subclass, Extract Superclass, Inline Method
Object Size	[S54]	Inline Temp
Improve general quality	[S54]	renaming of files and functions, splitting of long files, moving of functions from one module to another, conversion of macros to inline functions, changing of data type
Improve maintainability and reusability	[S54]	Removal of internal and external code clones Merging of different implementations and realization using conditional compilation. Reduction of the scale and complexity of functions.
Improve SPL design	[S15]	Remove optional component, Make optional component a core component, Make the one variant a core component and remove other variants, Remove the unused variants, Split off an optional component

Table 17
Summary of metrics to evaluate the SPL opportunities of refactoring, adapted from Critchlow et al. [S15], and Ribeiro et al. [S50].

Metric	Ref.	Description
PSU	[S15]	Provided Service Utilization
RSU	[S15]	Required Service Utilization
LOC	[S50]	The total number of lines in a module or function
CDC	[S50]	Concern Diffusion over Components (number of components that implements a concern)
CDLOC	[S50]	Concern Diffusion over Lines of Code
NCC	[S50]	Number of Concerns per Component
VS	[S50]	Vocabulary Size
CBC	[S50]	Total Coupling Between Components
DIT	[S50]	Depth of Inheritance Tree

and AO. Using Change impact analysis and modularity analysis, the results show that the combined use of components and aspects contributes to design stability when an SPL evolves. The study is partially one of the few empirical studies that evaluate state-of-the-art techniques for SPL design and implementation.

Heider et al. [S25] present a simulation tool for exploring the effects of product line evolution on model complexity and maintenance effort. The simulation considers quantitative data of product lines (e.g., size, dependencies in models) and they experiment with different evolution profiles.

The most interesting approaches try to establish advantages and disadvantages of the diverse techniques described in this Section from the point of view of the SPL refactoring and improvement. Empirical evidence has been presented favoring a mixed approach of components/AOP against pure OO or AOP options [S59]. More effort is required to assess recent FOP proposals (themselves being a combination of semi-automated composition and physical separation of concerns).

6.3. Tool summary for product line refactoring

Finally, in Table 18, a summary of the tools oriented to evolving an SPL is presented. Once the SPL is built, their dependences can be discovered using tools on Formal Concept Analysis. Other tools focus mainly on code, with the aim, for example, of transforming it into an Aspect Oriented version. However, there are practically no refactoring tools specifically designed for the needs of legacy system reengineering or SPL refactoring. Though conventional refactoring tools can be used, an effort should be made to adapt these tools to the particularities of the SPL paradigm, and in particular to the existing commercial or open source SPL implementation and configuration tools (the mentioned Pure::Variants,[3] Ahead, FeatureIDE, etc.).

7. Conclusions and research opportunities summary

A systematic mapping has been conducted on the reengineering of (families of) legacy systems into software product lines and on the refactoring of existing SPLs. We considered both subfields of a major field we named under the unifying expression *SPL oriented evolution*. The main goal was to delimit which topics have been investigated in the field and what the orientation of each approach is, from simply manual guidelines to specific techniques for reengineering/refactoring code or models (research question RQ1.1). We have studied the availability of tools, especially with real implantation in industry, and

[3] http://www.pure-systems.com/.

Table 18
Summary of tools used in SPL related refactoring papers.

Ref.	Tool	Main utility	Availability	Observations
[S23]	Concept Explorer	Formal Concept Analysis	Open source	Generic tool, requires preparation of data
[S3][S10]	FlipEX	Aspect oriented	Eclipse plug-in	Wizard to transformations
[S18]	*Reflexion* tools	Reengineering *Reflexion* Method	No details	
[S22]	Alloy Analyzer	Checking correctness of model refactorings	Free, MIT license	Formal language utilitiy
[S15]	ARCHMETRIC ARCHREFACTOR	service utilization metrics	No details (?)	Predefined refactoring strategies.

the empirical evidence obtained by using specific SPL metrics (research question RQ1.2). Proposals based on formal methods were considered a priori ideal candidates, though only a few works follow this line, so it remains an open issue. This connects with the last goal of the mapping study: the detection of research opportunities in the field (research question RQ2).

The first step was the classification of the papers selected in both subfields, guided by three main challenges: the overall process to conduct the migration to and the evolution of an SPL; model extraction or refactoring (including feature models); and code reengineering or refactoring techniques (in AOP, FOP, and conventional OOP variants). It is worth mentioning that model/code reengineering and refactoring techniques are considered globally on some occasions.

Once the studies were grouped, the state of the art, the list of existing tools and the diverse degrees of validation of each proposal were assessed and the open issues identified. Table 19 summarizes, in two columns, the state of the art and the open issues for each aspect under consideration. In each aspect we have identified techniques validated by case studies (most of them) and by industry applications. Empirical data have been collected in a few case studies. Formal validation is only found in the simplest task: the refactoring of feature variability models or SPL formal definitions. The main aspects and pending challenges are discussed and highlighted in the next paragraphs.

Methodology and process

Reengineering methods, tools and metrics are a good starting point to analyze legacy code and discover the features included in its different parts. The works in SPL introduction and evolution processes (initially promoted by the SEI and IESE organizations) provide *well established frameworks* to guide reengineering tasks. However, attention has recently been given to the *adaptation of agile processes*. It is too early to see the influence of these works and we think that the currently available information is still neither conclusive nor detailed enough, so it remains an open issue. The *impact of the migration* to the SPL paradigm on the global organization (how the productivity, time to market, or final product quality are increased) is not analyzed in the collected studies and this is another pending challenge.

Model extraction

We have found several techniques of *feature identification* in legacy systems. Some tools are also available to apply these techniques. However, more work should be done on the specific *techniques to analyze variability and reorganize the feature models* (FCA shows promising possibilities). Features are often known to interact with each other and, in consequence, one of the problems is to detect not only the features but also their interdependencies and hierarchy in the original code. An integration of these techniques in a *guided process and workbench of feature model extraction and optimization* is a pending challenge.

Feature extraction from legacy code

The *identification and extraction of components* appropriate to becoming product line assets have also been contemplated. Many of these works include empirical data, in an attempt to measure the suitability of these components to be reused (or reengineered) as product line assets. The choice of the *best target language for code reengineering and refactoring* is an open issue. Some works claim that AOP separation of concerns is the best way to implement features. Others have detected problems inherent to the AOP languages and propose combining different paradigms (components, *mixin* layers, AOP, etc.). On the other hand, using conventional languages, such as Java or C# (the partial class mechanisms of C# natively allow the separation of code), as target languages can help in the expansion of the SPL paradigm to a greater number of software companies. This can be achieved by devising efficient annotations that identify features and their dependencies in the code (*virtual separation of concerns,* as in CIDE), or separating code into some form of packages (*physical separation*) which can be managed by tools such as FeatureIDE [39] for Java.

Empirical analysis, guidance and assessment

Concerning the reengineering block, *empirical data* have been collected in some case studies to assess the quality of the resulting assets. The *definition of metrics* intended for the suitability of existing components as SPL assets or for the quality evaluation of the extracted components for the new SPL is frequent. A set of tools are available, although they suffer a *lack of integration with specific SPL suites*. This constitutes a research opportunity and a remaining challenge: We believe that important efforts still have to be made on *integration and standardization* of metrics in the near future.

Table 19
Summary of the state of the art, tools, and validation level together with the detected open issues and research opportunities in SPL oriented evolution.

Subfield/aspect	State of the art, tools, and validation	Open issues and research opportunities
Reengineering legacy systems		
• Methodology and process	• Classical SEI/IESE method, • Abundance of industry experiences • OAR [S56],MAP [S57], RE-PLACE [S7] methods • Agile process [S9],[S74]	• Empirical systematic evaluation of the improvements in process and products • Agile processes vs. classical processes assessment
• Model extraction	• Reengineering tools allow the identification and extraction of features from legacy code • RECON [25], FEAT [26], CIDE [S39] tools, freely available	• Integration of tools in an SPL oriented workbench (to convert detected features into feature models)
• Feature extraction from legacy code (OOP)	• Reengineering techniques and tools are used with conventional OOP languages Java/C++ (virtual separation of features) • Tools: DMS [S2], XVCL [S66], XFIG [S55]	• Evaluation and comparison of virtual and physical separation of features • Evaluation and comparison of AOP/AspectJ pure approach vs. FOP Java (C#) compositional approaches • Refactoring catalog integration and unification
• Feature extraction from legacy code (FOP)	• Extraction of Java extended code (physical separation of features) • FOP Refactoring catalog [S44] • Ahead [38] and FeatureIDE [39] free tools	
• Feature extraction from legacy code (AOP)	• AOP Refactoring catalog [S5][S3][S10] • Tools: FLiPEx plug-in [S3] [S10], CIDE [S39]	
• Empirical analysis, guidance and assessment	• Metrics proposed and applied in OOP context [S35][S36], and AOP context [S51]	• Normalization and classification of the metrics • More empirical evidence is required to evaluate the OOP/AOP/FOP alternatives
SPL Refactoring		
• SPL Model Refactoring	• Formal techniques (FCA, Alloy) guarantee that modified feature models are correct • Alloy Analyzer [S22], Concept Explorer [S42]	• Integration of these techniques and tools with code oriented tools to guide code refactoring
• SPL Code Refactoring	• A set of metrics to analyze SPLs and associated refactoring • FLiPEx plug-in [S3] [S10], ARCHMETRIC, ARCHREFACTOR [S15]	• More effort on specific refactoring tools (better if integrated with SPL IDE tools)

Model and Code SPL Refactoring

The refactorings defined (in a broad sense) to achieve the evolution should be cataloged and included in a suitable *refactoring/reengineering tool* specifically adapted to that problem. All the proposed code transformations can be categorized into three promising groups: pure product line refactoring, and FOP or AOP legacy reengineering. The elaboration of a *unified catalog of code refactorings* related with *SPL oriented evolution* is one of the topics incorporated into our agenda of immediate work.

Closely related to refactoring is the *collection of metrics* we have found in several contexts, in particular for the evaluation of the quality improvement of an evolved SPL after a refactoring process. Again, a precise categorization (and identification of overlapping of similar metrics) is missing. A similar concept to the well-known *Smell* concept (Design Smell, Code Smell, Bad Smell) in the context of SPL oriented evolution could benefit the detection of product line refactoring opportunities. Our intention is to connect the metric catalog with the recommended refactorings, obtaining a sort of *cookbook* for guiding all the evolution steps on the field of SPL oriented evolution.

Finally, some advances in the *formal definition of refactoring* are achieved, especially in the feature modeling and FOP contexts. However, these are rather limited. More work in this type of formalism is required to find the most suitable for integration with recent works on the formal definitions of SPLs.

Acknowledgments

This work has been funded by the Spanish MICIINN through the TIN2008-05675 project.

Appendix

See Table A.1.

Table A.1
The selected studies .

[S1]	S. A. Ajila, "Reusing base product features to develop product line architecture", in Information Reuse and Integration, Conf, 2005. IRI -2005 IEEE International Conference on., 2005, pp. 288–293.
[S2]	R. L. Akers, I. D. Baxter, M. Mehlich, B. J. Ellis, and K. R. Luecke, "Case study: Re-engineering C++ component models via automatic program transformation", Information and Software Technology, vol. 49, no. 3, pp. 275–291, 2007.
[S3]	V. Alves, F. Calheiros, V. Nepomuceno, A. Menezes, S. Soares, and P. Borba, "FLiP: Managing Software Product Line Extraction and Reaction with Aspects", in Software Product Line Conference, 2008 12th International, 2008, pp. 354–354.
[S4]	V. Alves, R. Gheyi, T. Massoni, U. Kulesza, P. Borba, and C. Lucena, "Refactoring product lines", in Generative programming and component engineering — GPCE '06, Proceedings of the 5th international conference on, 2006, p. 201.
[S5]	V. Alves, P. M. Jr, L. Cole, P. Borba, and G. Ramalho, "Extracting and Evolving Mobile Games Product Lines", in Software Product lines, Springer Verlag, Berlin, 2005.
[S6]	J. Bartholdt and D. Becker, "Re-engineering of a hierarchical product line", in Proceedings-15th International Software Product Line Conference, SPLC 2011, 2011, pp. 232–240.
[S7]	J. Bayer, J.-F. Girard, M. Würthner, J.-M. DeBaud, and M. Apel, "Transitioning Legacy Assets to a Product Line Architecture", in Software Engineering — ESEC/FSE '99, 1999, vol. 1687, pp. 446–463.
[S8]	C. Berger, H. Rendel, and B. Rumpe, "Measuring the Ability to Form a Product Line from Existing Products", in VAMOS, 2010.
[S9]	J. Bosch and P. M. Bosch-Sijtsema, "Introducing agile customer-centered development in a legacy software product line", Software: Practice and Experience, no. April, pp. 871–882, 2011.
[S10]	F. Calheiros, P. Borba, S. Soares, and V. Nepomuceno…, "Product line variability refactoring tool", in 1st Workshop on Refactoring Tools, Berlin (2007), 2007, pp. 33–34.
[S11]	F. Chen, S. Li, H. Yang, C.-H. Wang, and W. Cheng-Chung Chu, "Feature analysis for service-oriented reengineering", in Asia-Pacific Software Engineering Conference, 2005. APSEC '05. 12th, 2005, pp. 201–208.
[S12]	C.-C. Chiang and R. Y. Lee, "Developing tools for reverse engineering in a software product-line architecture", in Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, IRI-2004, 2004, pp. 42–47.
[S13]	J. M. Conejero, J. Hernández, and E. Jurado, "Early Analysis of Modularity in Software Product Lines", in 21st International Conference on Software Engineering and Knowledge Engineering, 2009, pp. 721–736.
[S14]	M. V. Couto, M. T. Valente, and E. Figueiredo, "Extracting software product lines: A case study using conditional compilation", in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2011, pp. 191–200.
[S15]	M. Critchlow, K. Dodd, J. Chou, and A. V. D. Hoek, "Refactoring product line architectures", in International Workshop on Refactoring: Achievements, Challenges, Effects (REFACE03), 2003.
[S16]	A. Dabholkar and A. Gokhale, "Middleware Specialization for Product-Lines Using Feature-Oriented Reverse Engineering", in Seventh International Conference on Information Technology, 2010, 2010, pp. 696–701.
[S17]	J.-M. DeBaud and J.-F. Girard, "The Relation Between the Product Line Development Entry Points and Reengineering", in Second International ESPRIT ARES Workshop on Development and Evolution of Software Architectures for Product Families, 1998, pp. 132–139.
[S18]	P. Frenzel, R. Koschke, A. P. J. Breu, and K. Angstmann, "Extending the Reflexion Method for Consolidating Software Variants into Product Lines", in Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on, 2007, pp. 160–169.
[S19]	D. Ganesan and J. Knodel, "Identifying Domain-Specific Reusable Components from Existing OO Systems to Support Product line Migration", in R2PL 2005—Proceedings of the First International Workshop on Reengineering Towards Product Lines, 2005.
[S20]	Y. Ghanam and F. Maurer, "Extreme Product Line Engineering–Refactoring for Variability: A Test-Driven Approach", in Agile Processes in Software Engineering and Extreme Programming, 2010, pp. 43–57.
[S21]	R. Gheyi, T. Massoni, and P. Borba, "Algebraic laws for feature models", Journal of Universal Computer Science, vol. 14, no. 21, pp. 3573–3591, 2008.
[S22]	R. Gheyi, T. Massoni, and P. Borba, "Automatically checking feature model refactorings", Journal of Universal Computer Science, vol. 17, no. 5, pp. 684–711, 2011.
[S23]	B. Graaf, S. Weber, and A. V. Deursen, "Migrating Supervisory Control Architectures Using Model Transformations", in Software Maintenance and Reengineering (CSMR 2006), 10th European Conf., 2006, pp. 151–160.
[S24]	C. H. B. Hansen K.M., "Component reengineering workshops: A low-cost approach for assessing specific reengineering costs across product lines", in Proceedings of the European Conference on Software Maintenance and Reengineering, CSMR, 2004, vol. 8, pp. 154–162.
[S25]	W. Heider, R. Froschauer, P. Grünbacher, R. Rabiser, and D. Dhungana, "Simulating evolution in model-based product line engineering", Information and Software Technology, vol. 52, no. 7, pp. 758–769, Jul. 2010.
[S26]	A. Hubaux, P. Heymans, and D. Benavides, "Variability modelling challenges from the trenches of an open source product line re-engineering project", in Proceedings-12th International Software Product Line Conference, SPLC 2008, 2008, pp. 55–64.
[S27]	A. Hubaux, P. Heymans, and H. Unphon, "Separating variability concerns in a product line re-engineering project", in Proceedings of the 2008 AOSD workshop on Early aspects EA 08, 2008, pp. 1–8.
[S28]	I. John, "Capturing Product Line Information from Legacy User Documentation", in Software Product Lines, Springer, 2006, pp. 127–159.
[S29]	I. John, "Integrating Legacy Documentation Assets into a Product Line", in Revised Papers from the 4th International Workshop on Software Product-Family Engineering, 2002, pp. 113–124.
[S30]	G. Jun, D. Eryu, and L. Bin, "Feature-oriented re-engineering using product line approach", in 2nd International Conference on Information Science and Engineering, ICISE2010-Proceedings, 2010, pp. 255–260.
[S31]	K. C. Kang, M. Kim, J. Lee, and B. Kim, "Feature-oriented re-engineering of legacy systems into product line assets–a case study", in Software Product Lines, 2005, pp. 45–56.
[S32]	K. Kim, H. Kim, and W. Kim, "Building Software Product Line from the Legacy Systems 'Experience in the Digital Audio and Video Domain'", in Software Product Line Conference, 2007. SPLC 2007. 11th International, 2007, pp. 171–180.
[S33]	J. Knodel and D. Muthig, "Analyzing the Product Line Adequacy of Existing Components", in R2PL 2005—Proceedings of the First International Workshop on Reengineering Towards Product Lines, 2005.
[S34]	J. Knodel et al., "Asset Recovery and Incorporation into Product Lines", in Proceedings of the 12th Working Conference on Reverse Engineering, 2005, p. 120.

Table A.1
The selected studies (contd.).

[S35]	R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi, "A Case Study in Refactoring a Legacy Component for Reuse in a Product Line", in International Conference on Software Maintenance (ICSM'05), 21st IEEE, 2005, pp. 369–378.
[S36]	R. Kolb, D. Muthig, T. Patzke, and K. Yamauchi, "Refactoring a legacy component for reuse in a software product line: a case study", J. Softw. Maint. Evol., vol. 18, no. 2, pp. 109–132, 2006.
[S37]	C. Kästner, S. Apel, and D. Batory, "A Case Study Implementing Features Using AspectJ", in Software Product Line Conference, 2007. SPLC 2007. 11th International, 2007, pp. 223–232.
[S38]	C. Kästner, S. Apel, and M. Kuhlemann, "A model of refactoring physically and virtually separated features", in GPCE'09 - Proceedings of the 8th International ACM SIGPLAN Conference on Generative Programming and Component Engineering, 2009, pp. 157–166.
[S39]	C. Kästner, M. Kuhlemann, and D. S. Batory, "Automatic Feature Oriented refactoring of legacy applications", in Workshop on Refactoring Tools, WRT 2007, 2007, pp. 62–63.
[S40]	H. Lee, H. Choi, K. Kang, D. Kim, and Z. Lee, "Experience Report on Using a Domain Model-Based Extractive Approach to Software Product Line Asset Development", in Formal Foundations of Reuse and Domain Engineering (ICSR 2009), 2009, vol. 5791, pp. 137–149.
[S41]	J. Liu, D. Batory, and C. Lengauer, "Feature oriented refactoring of legacy applications", in international conference on Software engineering - ICSE '06, Proceeding of the 28th, 2006, p. 112.
[S42]	F. Loesch and E. Ploedereder, "Optimization of Variability in Software Product Lines", in Software Product Line Conference, 2007. SPLC 2007. 11th International, 2007, pp. 151–162.
[S43]	F. Loesch and E. Ploedereder, "Restructuring Variability in Software Product Lines using Concept Analysis of Product Configurations", in Software Maintenance and Reengineering, 2007. CSMR '07. 11th European Conference on, 2007, pp. 159–170.
[S44]	R. E. Lopez-Herrejon, L. Montalvillo-Mendizabal, and A. Egyed, "From requirements to features: An exploratory study of feature-oriented refactoring", in Proceedings-15th International Software Product Line Conference, SPLC 2011, 2011, pp. 181–190.
[S45]	A. Olszak and B. N. Jørgensen, "Remodularizing Java programs for comprehension of features", in First International Workshop on Feature-Oriented Software Development - FOSD '09, 2009, p. 19.
[S46]	L. O'Brien, F. Hansen, R. Seacord, and D. Smith, "Mining and managing software assets", in Software Technology and Engineering Practice (STEP'02), 2002, pp. 82–90.
[S47]	I. Pashov and M. Riebisch, "Using feature modeling for program comprehension and software architecture recovery", in 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004, pp. 406–417.
[S48]	X. Peng, Y. Yu, and W. Zhao, "Analyzing evolution of variability in a software product line: From contexts and requirements to features", Information and Software Technology, vol. 53, no. 7, pp. 707–721, Jan. 2011.
[S49]	M. Ribeiro and P. Borba, "Recommending refactorings when restructuring variabilities in software product lines", in Proceedings of the 2nd Workshop on Refactoring Tools, WRT '08, in conjunction with the Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA, 2008.
[S50]	M. Ribeiro and P. Borba, "Improving Guidance when Restructuring Variabilities in Software Product Lines", in Software Maintenance and Reengineering, 2009. CSMR '09. 13th European Conference on, 2009, pp. 79–88.
[S51]	D. Saraiva, L. Pereira, T. Batista, and F. Delicat, "Architecting a Model-Driven Aspect-Oriented Product Line for a Digital TV Middleware: A Refactoring Experience", in Software Architecture: 4th European Conference , ECSA 2010,, 2010, pp. 166–181.
[S52]	I. Savga and F. Heidenreich, "Refactoring in feature-oriented programming: Open issues", in Workshop on Modularization, Composition, and Generative Techniques for Product Line Engineering, 2008, pp. 41–46.
[S53]	J. Savolainen, I. Oliver, V. Myllärniemi, and T. Männistö, "Analyzing and re-structuring product line dependencies", in Proceedings - International Computer Software and Applications Conference, 2007, vol. 1, pp. 569–572.
[S54]	N. Siegmund, M. Kuhlemann, M. Pukall, and S. Apel, "Optimizing Non-functional Properties of Software Product Lines by means of Refactorings", in VaMoS'10, 2010, pp. 115–122.
[S55]	D. Simon and T. Eisenbarth, "Evolutionary Introduction of Software Product Lines", in Software Product Lines, 2002, vol. 2379, pp. 1–14.
[S56]	D. B. Smith, L. O'Brien, and J. Bergey, "Using the Options Analysis for Reengineering (OAR) Method for Mining Components for a Product Line", in Software Product Lines, second International Conference on, 2002, pp. 316–327.
[S57]	C. Stoermer and L. O'Brien, "MAP - Mining Architectures for Product Line Evaluations", in IEEE/IFIP Working Conference on Software Architecture, 2001, pp. 35–44.
[S58]	R. Stoiber, S. Fricker, M. Jehle, and M. Glinz, "Feature Unweaving: Refactoring Software Requirements Specifications into Software Product Lines", in 18th IEEE International Requirements Engineering Conference, 2010, pp. 403–404.
[S59]	L. P. Tizzei and J. Lee, "An Aspect-oriented View to Support Feature-oriented Reengineering", in Workshop on Aspect-Oriented Modeling (AOM'10) at MODELS, 2010.
[S60]	L. P. Tizzei, M. Dias, C. M. F. Rubira, A. Garcia, and J. Lee, "Components meet aspects: Assessing design stability of a software product line", Information and Software Technology, vol. 53, no. 2, pp. 121–136, 2011.
[S61]	S. Trujillo, D. Batory, and O. Diaz, "Feature refactoring a multi-representation program into a product line", in Proceedings of the 5th international conference on Generative programming and component engineering, 2006, pp. 191–200.
[S62]	N. Weiderman, J. Bergey, D. Smith, and S. Tilley, "Can Legacy Systems Beget Product Lines?", in Second International ESPRIT ARES Workshop, 1998.
[S63]	Y. Wu, Y. Yang, X. Peng, C. Qiu, and W. Zhao, "Recovering object-oriented framework for software product line reengineering", in Proceedings of the 12th international conference on Top productivity through software reuse, 2011, pp. 119–134.
[S64]	Y. Xue, Z. Xing, and S. Jarzabek, "Understanding feature evolution in a family of product variants", in Working Conference on Reverse Engineering, WCRE, 2010, pp. 109–118.
[S65]	Y. Xue, "Reengineering legacy software products into software product line based on automatic variability analysis", in Proceedings of International Conference on Software Engineering, 2011, pp. 1114–1117.
[S66]	W. Zhang, S. Jarzabek, N. Loughran, and A. Rashid, "Reengineering a PC-based system into the mobile device product line", in Sixth International Workshop on Principles of Software Evolution, 2003. Proceedings., 2002, pp. 149–160.
[S67]	H. P. Breivold, S. Larsson, and R. Land, "Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies", 34th Euromicro Conference Software Engineering and Advanced Applications, pp. 232–239, Sep. 2008.
[S69]	A. Gruler, M. Leucker, K. Scheidemann, and T. Universit, "Calculating and Modeling Common Parts of Software Product Lines", in 12th International Software Product Line Conference, 2008, pp. 203–212.

Table A.1
The selected studies (contd.).

[S69]	A. Lozano, "An overview of techniques for detecting software variability concepts in source code", in Advances in Conceptual Modeling. Recent Developments and new directions, 2011, pp. 141–150.
[S70]	M. Pinzger et al., "Architecture recovery for product families", in Software Product-Family Engineering, 2004, pp. 332–351.
[S71]	G. Raghavan, "Improving software quality in product families through systematic reengineering", in 7th European Conference on Software Quality, 2002, pp. 90–99.
[S72]	K. Romanovsky, D. Koznov, and L. Minchin, "Refactoring the Documentation of Software", in Software Engineering Techniques, 2011, pp. 158–170.
[S73]	W. L. Scherlis, "Structural Views, Structural Evolution, and Product Families", in Development and Evolution of Software Architectures for Product Families, 1998, pp. 235–240.
[S74]	G. Zhang, L. Shen, X. Peng, Z. Xing, and W. Zhao, "Incremental and Iterative Reengineering towards Software Product Line : An Industrial Case Study", in 2011 International Conference On Software Maintenance, 2011, pp. 418–427.

References

[1] F. van der Linden, K. Schmid, E. Rommes, Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering, Springer, 2007, p. 333.
[2] SEI, A Framework for Software Product Line Practice, Version 5.0. [Online]. Available: Accessed: 11-Nov-2011.
[3] J. Bosch, Design & Use of Software Architectures. Adopting and Evolving a Product-Line Approach, Addison-Wesley, 2000.
[4] J. Bayer, J.-F. Girard, M. Würthner, J.-M. DeBaud, M. Apel, Transitioning legacy assets to a product line architecture, in: Software Engineering — ESEC/FSE'99, vol. 1687, 1999, pp. 446–463.
[5] M.A. Laguna, B. González-Baixauli, J.M. Marqués, Seamless development of software product lines, in: Proceedings of the 6th international conference on Generative programming and component engineering - GPCE '07, 2007, pp. 85–94.
[6] M. Laguna, J.M. Marqués, UML support for designing software product lines: the package merge mechanism, Journal of Universal Computer Science 16 (17) (2010) 2313–2332.
[7] R. Marticorena, C. López, Y. Crespo, J. Pérez, Refactoring generics in JAVA: a case study on Extract Method, in: 14th European Conference on Software Maintenance and Reengineering, CSMR 2010, 2010, pp. 217–226.
[8] R. Marticorena, C. López, J. Pérez, Y. Crespo, Assisting refactoring tool development through refactoring characterization, in: 6th International Conference on Software and Data Technologies, ICSOFT 2011, vol. 2, 2011, pp. 232–237.
[9] B.A. Kitchenham, T. Dyba, M. Jorgensen, Evidence-based software engineering, in: Proceedings. 26th International Conference on Software Engineering, pp. 273–281.
[10] B. Kitchenham, S. Charters, Guidelines for performing systematic literature reviews in software engineering, T.R. Keele University, 2007.
[11] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: 12th International Conference on Evaluation and Assessment in Software Engineering, 2008, pp. 71–80.
[12] B. Kitchenham, D. Budgen, O. Pearl Brereton, Using mapping studies as the basis for further research – A participant-observer case study, Information and Software Technology 53 (6) (2011) 638–651.
[13] W.F. Opdyke, Refactoring Object-Oriented Frameworks. Ph.D. Thesis, University of Illinois at Urbana-Champaign, IL, USA, 1992.
[14] M. Fowler, K. Beck, J. Brant, W. Opdyke, D. Roberts, Refactoring: Improving the Design of Existing Code, Addison Wesley, 1999.
[15] J. Pérez, C. López, N. Moha, T. Mens, A Classification Framework and Survey for Design Smell Management, Informe Técnico 2011/01, 2011.
[16] J. Kerievsky, Refactoring to Patterns, Addison-Wesley, 2004.
[17] R.S. Arnold, Software restructuring, Proceedings of the IEEE 77 (4) (1989) 607–617.
[18] E. Chikofsky, J. Cross, Reverse engineering and design recovery: a taxonomy, IEEE Software 7 (1) (1990) 13–18.
[19] N.H. Madhavji, J. Fernandez-Ramil, D. Perry, Software Evolution and Feedback: Theory and Practice, John Wiley & Sons, 2006.
[20] C.L. Nehaniv, P. Wernick, Introduction to software evolvability, in: International Workshop on the Principles of Software Evolution, IWPSE, 2007.
[21] H.P. Breivold, I. Crnkovic, M. Larsson, A systematic review of software architecture evolution research, Information and Software Technology 54 (1) (2012) 16–40.
[22] C. Stoermer, L. O'Brien, MAP - Mining Architectures for Product Line Evaluations, in: IEEE/IFIP Working Conference on Software Architecture, 2001, pp. 35–44.
[23] L. O'Brien, F. Hansen, R. Seacord, D. Smith, Mining and managing software assets, in: Software Technology and Engineering Practice, STEP'02, 2002, pp. 82–90.
[24] IESE, Fraunhofer-Institut für Experimentelles Software Engineering. [Online]. Available: http://www.iese.fraunhofer.de. [Accessed: 20-Apr-2012].
[25] J. Bayer, J.-F. Girard, M. Würthner, J.-M. DeBaud, M. Apel, Transitioning legacy assets to a product line architecture, in: O. Nierstrasz, M. Lemoine (Eds.), Software Engineering — ESEC/FSE '99, vol. 1687, Springer, Berlin / Heidelberg, 1999, pp. 446–463.
[26] M. Shaw, What makes good research in software engineering? International Journal on Software Tools for Technology Transfer (STTT) 4 (1) (2002) 1-7-7.
[27] O. Dieste, A. Grimán, N. Juristo, Developing search strategies for detecting relevant experiments, Empirical Software Engineering 14 (5) (2008) 513–539.
[28] T. Dyba, T. Dingsoyr, G.K. Hanssen, Applying Systematic Reviews to Diverse Study Types: An Experience Report, in: First International Symposium on Empirical Software Engineering and Measurement, ESEM 2007, 2007, pp. 225–234.
[29] R. Wieringa, N.A.M. Maiden, N.R. Mead, C. Rolland, Requirements engineering paper classification and evaluation criteria: a proposal and a discussion, Requirements Engineering 11 (1) (2006) 102–107.
[30] P. Runeson, M. Skoglund, Reference-based search strategies in systematic reviews, in: 13th International Conference on Empirical Assessment & Evaluation in Software Engineering, 2009.
[31] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlson, B. Regnell, A. Wesslén, Experimentation in Software Engineering: An Introduction, Kluwer Academic, 2000.
[32] R. Kazman, S.J. Carrière, Playing detective: reconstructing software architecture from available evidence, Journal of Automated Software Engineering (April) (1999) 107–138.
[33] R. Stoiber, M. Glinz, Supporting stepwise, incremental product derivation in product line requirements engineering., in: VaMoS'10, 2010, pp. 77–84.
[34] N. Wilde, M. Buckellew, H. Page, V. Rajlich, L. Pounds, A comparison of methods for locating features in legacy software, Journal of Systems and Software 65 (2) (2003) 105–114.
[35] M.R. Robillard, G.C. Murphy, Concern graphs: finding and describing concerns using structural program dependencies, in: Proceedings of the 24th International Conference on Software Engineering, ICSE 2002, pp. 406–416.
[36] AMMA, ATL Development Tools, 2010. [Online]. Available: http://www.eclipse.org/m2m/atl/doc/. [Accessed: 20-Apr-2012].
[37] M. Glinz, Object-oriented modeling with ADORA, Information Systems 27 (6) (2002) 425–444.
[38] D. Batory, J.N. Sarvela, A. Rauschmayer, Scaling step- wise refinement, IEEE Transactions on Software Engineering 30 (6) (2004) 355–371.

[39] Database-Workgroup, FeatureIde: Eclipse-Plugin for Feature-Oriented Software Development, Otto-von-Guericke-Universität Magdeburg. [Online]. Available: http://www.fosd.de/fide. [Accessed: 20-Apr-2012].
[40] V. Basili, G. Calidera, D. Rombach, The goal/question/metric paradigm, in: J. Marciniak (Ed.), Encyclopedia of Software Engineering, vol. 1, John Wiley & Sons, 1994.
[41] S.A. Yevtushenko, System of data analysis: concept explorer, in: 7th National Conference on Artificial Intelligence, 2000, pp. 127–134.
[42] R. Milner, A Calculus for Communicating Processes, in: LNCS, vol. 92, Springer, 1980.
[43] G.C. Murphy, D. Notkin, K.J. Sullivan, Software reflexion models: bridging the gap between design and implementation, IEEE Transactions on Software Engineering 27 (4) (2001) 364–380.