The Journal of Systems and Software 122 (2016) 110–143

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Requirement-driven evolution in software product lines: A systematic mapping study

Leticia Montalvillo*, Oscar Díaz

University of the Basque Country (UPV/EHU), ONEKIN Research Group - Facultad de Informática - San Sebastián, Spain

ARTICLE INFO

Article history:
Received 11 December 2015
Revised 11 August 2016
Accepted 12 August 2016
Available online 21 August 2016

Keywords:
Systematic mapping study
Software product lines
Evolution

ABSTRACT

CONTEXT. Software Product Lines (SPLs) aim to support the development of a whole family of software products through systematic reuse of shared assets. As SPLs exhibit a long life-span, evolution is an even greater concern than for single-systems. For the purpose of this work, evolution refers to the adaptation of the SPL as a result of changing requirements. Hence, evolution is triggered by requirement changes, and not by bug fixing or refactoring.

OBJECTIVE. Research on SPL evolution has not been previously mapped. This work provides a mapping study along Petersen's and Kichenham's guidelines, to identify strong areas of knowledge, trends and gaps.

RESULTS. We identified 107 relevant contributions. They were classified according to four facets: evolution activity (e.g., identify, analyze and plan, implement), product-derivation approach (e.g., annotation-based, composition-based), research type (e.g., solution, experience, evaluation), and asset type (i.e., variability model, SPL architecture, code assets and products).

CONCLUSION. Analyses of the results indicate that "Solution proposals" are the most common type of contribution (31%). Regarding the evolution activity, "Implement change" (43%) and "Analyze and plan change" (37%) are the most covered ones. A finer-grained analysis uncovered some tasks as being under-exposed. A detailed description of the 107 papers is also included.

© 2016 Elsevier Inc. All rights reserved.

1. Introduction

Software Product Lines (SPLs) aim to support the development of a whole family of software products through systematic reuse of shared assets (Clements and Northrop, 2001). SPL engineering has gained considerable momentum. Companies such as Boeing, Bosch, General Motors, Philips or Siemens resort to SPLs to broaden their software portfolio, increase return on investment, shorten time to market, and improve software quality[1] (Clements and Northrop, 2001; van der Linden et al., 2007; Weiss, 2008). As the SPL domain matures, evolution concerns come into play (Bosch, 2002; Dhungana et al., 2008). Unfortunately, the term "evolution" has long been recognized as being overloaded with diverse matters (Bennett and Rajlich, 2000). For the purpose of this work, "evolution" refers to the adaptation of the SPL as *a result of changing SPL requirements*. From this perspective, evolution is triggered by requirement changes, and not so much by bug fixing or refactoring. Evolution happens as a result of SPLs moving from adoption to maturity. In their infancy, SPLs strive to fix defects. At adulthood, SPLs might have less defects, but their wider customer base more likely increases the chances for new functionality requests. Indeed, SPLs' long life-span makes evolution a top priority, yet far from being fully resolved (Botterweck and Pleuss, 2014). SPL characteristics that make evolution specially challenging include: (1) separation of development into Domain Engineering (DE) and Application Engineering (AE), (2) existence of assets of different types of variability and abstraction, and (3), high number of interrelations between assets (Mcgregor, 2003; Ajila and Kaba, 2008; Deelstra et al., 2005).

One of the first works addressing evolution in SPLs is Svahnberg and Bosch (1999). Svahnberg et al. analyze the life-span of two industrial SPLs, and classified SPL evolution according to common scenarios that arose during evolution ("requirement evolution", "architecture evolution", and "component evolution"). Thereafter, few efforts have been made to gather studies addressing this issue. Two exceptions are (Botterweck and Pleuss, 2014; Mcgregor, 2003). The most referenced work is McGregor's one who introduces basic evolution concepts and discusses practices that initiate, anticipate, control, and direct the evolution of SPL assets (Mcgregor, 2003). Botterweck and Pleuss (2014) present the most recent summary

* Corresponding author.
E-mail addresses: leticia.montalvillo@ehu.eus (L. Montalvillo), oscar.diaz@ehu.eus (O. Díaz).

[1] Refer to http://splc.net/fame.html for a list of successful SPL examples.

http://dx.doi.org/10.1016/j.jss.2016.08.053
0164-1212/© 2016 Elsevier Inc. All rights reserved.

on the topic. Authors provide an overview on three main issues: migration to SPLs, planning SPL evolution, and implementation of SPL evolution. None of the previous works systematically review the existing literature, and thus, they do not provide a complete coverage of the different topics.

A systematic mapping study is an evidence-based approach where existing works can be categorized, often giving a visual map of its results (Kitchenham and Charters, 2007; Petersen et al., 2008a). This work presents the outcome of such approach conducted for the literature on SPL evolution available up to July, 2015 which resulted in 107 primary studies. The overall research questions follow:

RQ1: What types of research have been reported, to what extent, and how is coverage evolving?

RQ2: Which product-derivation approach received most coverage, and how is coverage evolving?

RQ3: Which kind of SPL asset received more attention, and how is attention evolving?

RQ4: Which activities of the evolution life-cycle received most coverage, and how is this coverage evolving?

Answering RQ1 would allow us to assess maturity within the field, e.g., if research is limited to solution proposals or rather it takes a step forward and conducts some kind of validation, or even better, it evaluates the solution in industry. On the other hand, RQ2 would allow us to assess how SPL product derivation approaches are catching on. Next, RQ3 looks at *"the subject"* of evolution, i.e., the SPL asset being subject to change. This includes the variability model, the SPL architecture, code assets or SPL's products. Conversely, RQ4 looks at *"the verb"* of evolution, i.e., which evolution tasks authors have focused on (e.g., identify change, analyze change, implement change, verify change). In summary, the outcome of this study might help to identify trends, hotspots and gaps both in terms of "the verb" and "the subject" of SPL evolution. Also, a brief is provided for each of the 107 primary studies. Special effort is dedicated to arrange these studies within a fine-grained schema that might help newcomers to better pinpoint the area of interest.

The remainder of this paper is organized as follows. Section 2 provides an overview on SPLs, highlights what makes SPL evolution challenging, and points to previous mapping studies in the SPL field. Section 3 describes the systematic methodology used to conduct this mapping study. Section 4 provides an annotated bibliography that serves to map primary studies into a finer-grained classification of the evolution activities. Section 5 analyses the results of the mapping, and answers the RQs. Conclusions end the paper.

2. Background

This section provides an overview on SPLs, highlights what makes SPL evolution challenging, and points to previous mapping studies in the SPL field.

2.1. A brief on SPLs

SPLs aim to support the development of a whole family of software products through systematic reuse of shared assets (Clements and Northrop, 2001). These assets give support to different stages of the SPL production process. The asset list includes variability models (i.e., allowed variants to be exhibited by the SPL products, a.k.a. features), architecture (i.e., high-level description of the main modules involved and their connections), software components, class libraries, code snippets or at a higher description level, models as in model-driven SPLs. It might also include requirement documents, plans, test cases, process descriptions, product configurations, and trace documents. These assets are handled along two interrelated processes. During Domain Engineering (DE), the scope and variability of the SPL are defined, and reusable assets are developed. During Application Engineering (AE), products are derived using these assets by resolving variability (Pohl et al., 2005). Hence, variability management is an SPL hallmark. SPL assets can be of different variabilities: *common* assets are present in all products, *variable* assets are present in some products, and *product-specific* assets are local to individual products.

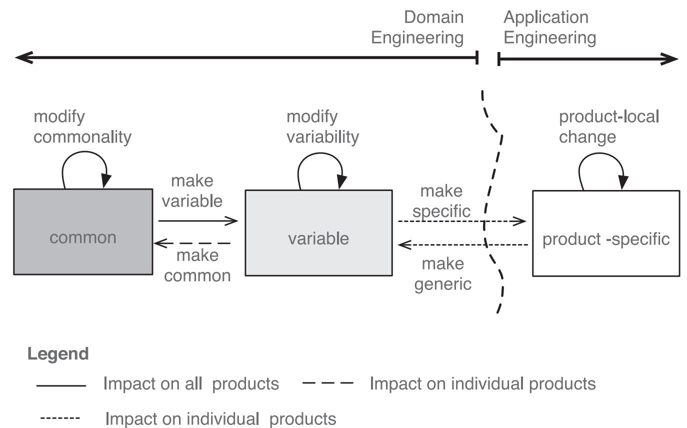

Fig. 2.1. Types of changes (based on (Botterweck and Pleuss, 2014; Schmid and Eichelberger, 2007)).

As any other software, SPLs are subject to evolution (Deelstra et al., 2005). Specifically, we conceive evolution as adaptation of the SPL to cope with *changing requirements*. This might happen in two different scenarios:

- during product derivation, new requirements emerge (a.k.a. reactive evolution). These requirements can be accounted for in two different places: within the product realm or within the core asset realm. The former implies the creation of product specific artifacts. Application engineers can use the core assets as basis for further development, or they can develop new assets from scratch. Second option is within the core asset realm. Here, requirements are tackled by domain engineers, and additions can benefit products other than the one generating the change.
- at any time, SPL engineers must be able to anticipate future needs (a.k.a. proactive evolution). This might lead to adapt core assets in such a way that the SPL is capable of accommodating the needs of product stakeholders in the shortest amount of time.

Previous scenarios involve **SPL changes**. Fig. 2.1 depicts the main types of changes along the lines of those proposed in Botterweck and Pleuss (2014); Schmid and Eichelberger (2007). Common functionality can be made variable if it should be excluded from some products. Usually, this requires changing the implementation (to make it variable), which then affects all existing products. Conversely, making a variable asset common influences at least those products that did not contain the asset before. Making a variable asset product-specific, or a product-specific asset generic, requires also to adapt individual products to hold or unhold the asset, respectively.

The bottom line is that SPL assets might be moved along "the variability spectrum": *common, variable* and *product-specific.* Common assets are present in all products, variable assets are present in some products, and product-specific assets are local to individ-

ual products. Moving along this spectrum is not straightforward due to SPL specifics, namely:

- **Large number of asset inter-dependencies.** The distinction between DE and AE introduces dependencies between products and the reusable assets used in their production. DE and AE have their own life-cycles and priorities. The urgency in releasing a product, fixing a bug, providing a new product release, or delivering a new feature may vary depending on whether the stakeholder is involved in DE or AE. Nevertheless, both parties need to be in sync to avoid SPL erosion (Deelstra et al., 2005).
- **Broad scope**. SPLs aim to build a family of products. Hence, the volume and likelihood of asset coupling is potentially larger than if the focus were on a single product.
- **Large life-span**. SPLs are long-term investments. This lengthy life-span should encourage a more effective control over SPL evolution in order to avoid SPL decay (van Gurp and Bosch, 2002).

A final remark. Terminology was particularly elusive in this study. In the SPL literature, the term "evolution" can denote a broad range of concerns: migrating legacy systems into SPLs (e.g., Laguna and Crespo, 2013), refactoring (e.g., Laguna and Crespo, 2013) or bug-fixing (e.g., Ribeiro and Borba, 2008; Schulze et al., 2013), to name a few. This is not specific of the SPL literature but it has long been recognized for software engineering in general (Bennett and Rajlich, 2000). The term "maintenance" tends to be predominantly used to describe activities aiming at preventing software from failing to deliver the intended functionalities. In the same vein, SEBOK defines maintainability as "the probability that a system or system elements can be repaired in a defined environment within a specified period of time" SEBOK. It can be noticed a bias towards the use of the term maintenance in relation with "failure" and "repair". From this perspective, maintenance predominantly aims at preserving functionality. By contrast, we conceive "evolution" not so much as a repairing action, but as an enhancement in the system's capabilities. Here, stakeholders (rather than bugs) tend to be the main triggers of evolution. This distinction is aligned with the way software modifications are classified by Kitchenham et al. (1999). Rather than using Swanson's classification of maintenance activities based on intention (i.e., corrective, adaptive, and perfective) (Swanson, 1976), Kitchenham et al. propose to categorize the modifications in terms of activities performed: activities to make corrections (i.e., existence of discrepancies between the expected behavior of a system and the actual behavior) *versus* activities to make enhancements (i.e., existence of desires to somehow change the current behavior of the system). For the purpose of this work, we use the term "evolution" to denote these enhancement activities, would these be modifying the scope, the commonality, the variability or the products of an SPL. We then leave out activities such as SPL migration (Breivold et al., 2008; Laguna and Crespo, 2013), SPL bad-smell detection (Abdelmoez et al., 2004; Devine et al., 2014; Loesch and Ploedereder, 2007; Bertran et al., 2010; Padilha et al., 2014; Vale et al., 2014), SPL refactoring (Alves et al., 2008, 2006; Ribeiro and Borba, 2008; Schulze et al., 2012, 2013) or SPL bug fixing (Krishnan et al., 2011, 2013). At adulthood, SPL is exposed to a wider customer base and hence, the pressure for new functionality increases. As pointed out by Singer, "a corrective activity may require only the ability to locate faulty code and make localized changes, whereas an enhancement activity may require a broad understanding of a large part of the product" (Singer, 1998). Our research questions are headed for assessing the types and coverage of these "enhancement activities".

2.2. Related mapping studies

We conducted a Scopus[2] search for mapping studies in SPLs published from 2010. The following search string was used:

("software product line" OR "SPL") AND ("systematic literature review" OR "systematic review" OR "research review" OR "systematic overview" OR "mapping study")

We identified six relevant papers that overlap with our interests (see Table 1). For quality attributes in SPLs, Montagud et al. (2012) found 165 measures proposed in the literature. This figure is broken down along the SPL life-cycle phase in which the measures are applied: *Requirements* (9%), *Design* (67%), *Realization* (4%), *Testing* (3%), *Application domain phase* (7%), and, most important here, the *Evolution stage* (10%). The latter is based on the insights of a single paper: (Ajila and Dumitrescu, 2007).

Laguna and Crespo (2013) address the reengineering of legacy systems into SPLs. Here, the term *evolution* is understood as the effect of migrating a set of related products, probably created by clone-and-own operation, into an SPL where reusable assets are obtained by refactoring existing products. Our focus is not so much in how SPLs are created by reengineering existing products, but SPLs' assets evolution. Indeed, studies of Laguna et al. present no overlap with our primary studies. Though refactoring is certainly a trigger for evolution, we are more interested in how SPL engineers accommodate new functionality. This makes Risk Management (RM) a topic of special interest. The mapping study conducted by Lobato et al. (2013) identifies RM activities and practices in SPLs. Some practices tackle the evolution of SPLs. For instance, the practice *SPL variability* acknowledges that "the product variability must be considered when evolving the architecture". However, SPL evolution does not appear as a first-class activity but is scattered among other steps (e.g., *SPL management, SPL variability, SPL testing*, etc.). By contrast, we move SPL evolution to the forefront, aiming to provide a broader overview of the different aspects involved, not limited to RM. Nevertheless, all the references concerning evolution were also included in our study.

Pereira et al. (2015) focus on SPL management tools. A classification facet is about the functionality cluster supported by the tool: *Planning* (i.e., means for collecting the data needed to define domain scope), *Modeling* (i.e., means for represents the domain scope), *Validation* (i.e., means for validating the domain), *Product configuration* (i.e., means for product derivation) and *Import/Export* facilities. The outcome provides the following distribution: *Planning* (34%), *Modeling* (85% of the tools support at least four of the functionalities), *Validation* (49% support at least three of the functionalities), *Product configuration* (83%) and *Import/Export* (71%). However, evolution as such is not explicitly considered but blurred behind other notions, mainly the *Validation* cluster which comprises functions for the inclusion of new requirements. It is not clear the extent to which tools give support to the evolution life-cycle (see later).

For consistency checking, Santos et al. (2015) undertook a mapping study for 24 primary studies. This work is certainly of interest for SPL evolution. Indeed, consistency checking aims at assuring that all SPL assets remain consistent with each other after some changes have been introduced: *model against source code* (25%), *model against model* (33%), or *model against specifications* (42%), where rates are those provided by this study. Our work extends beyond consistency checking to include other activities of the change life-cycle (Yau et al., 1978): identify change, analyze and plan change, implement change or verify change.

[2] http://www.scopus.com/.

Table 1
Related mapping studies.

Ref.	Year	Topic	Research Questions
(Montagud et al., 2012)	2012	Quality attribute	What quality attributes have been proposed for assessing the quality of software product lines? What measures have been proposed for assessing the quality of software product lines and how are they used?
(Laguna and Crespo, 2013)	2013	Migration	What approaches have been proposed on SPL oriented evolution and what is their focus and origin? Which challenges for SPL oriented evolution have been identified?
(Lobato et al., 2013)	2013	Risk management	Which risk management steps are suggested by the approaches? Which risks were identified and reported in SPLs? Which risk management activities and practices are adopted by the SPL approaches? What do the researchers commonly use to evaluate the identified risks? How do the stakeholders influence the identified risks?
(Pereira et al., 2015)	2014	Management tools	How many SPL management tools have been cited in the literature since 2000? What are the main characteristics of the tools? What are the main functionalities of the tools?
(Santos et al., 2015)	2015	Consistency checking	What kind of consistency checking activities have been performed in the literature? Can any trend on consistency checking be recognized in the research field? How do the existing approaches relate to each other?
(Heradio et al., 2016)	2015	Bibliometric analysis of SPL research	What are the most influential papers on SPL literature? Who are the most prolific authors? What journals, conferences, etc. have published the majority of the papers? How numerous is the SPL literature? How has paper publication been distributed over time? What are the main topics studied in the area? How has the interest in those topics evolved with time? What are the most impacting papers for a given topic along a certain period of time?

Finally, Heradio et al. (2016) perform the broadest mapping study on SPL research. Authors analyzed 20 years of the SPL literature (from 1995 to 2014), which involved above 2800 primary studies. Authors, resort to bibliometric analyses to: (1) identify the most influential publications on the SPL literature (based on received citations), (2) detect the most covered SPL “research topics” (in terms of published papers), and (3), determine how the interest in these research topics evolved over time. Main research topics are: *software architecture, automated analysis, feature modeling, software reuse, variability management, software quality, product derivation, domain engineering*, and *software design*. Regarding the evolution over time, authors ascertain that: (1) *software architecture* was the initial motor of research in SPLs; (2) work on *software reuse* has been essential for the development of the SPL research; and (3) *feature modeling* has been the most important topic for the last fifteen years, having the best evolution behavior in terms of number of published papers and received citations. From our perspective, it is worth highlighting that SPL evolution does not appear as a first-class topic, but included as part of *software reuse* and *software design*.

These studies can be considered good sources of information on their subjects. Yet, SPL evolution tends to be blurred behind other notions (e.g., migration, risk management, consistency checking, etc.). We aim at moving SPL evolution at the forefront by providing a deeper analysis along the lines of the change life-cycle stages (Yau et al., 1978).

3. Method

A Systematic Mapping Study (SMS) is an evidence-based form of secondary study. It provides a wide overview of a research area, to establish if research evidence exists on a topic, and provides an indication of the quantity of the evidence (Kitchenham and Charters, 2007). SMSs offer multiple benefits (Budgen et al., 2008). First, SMSs identify gaps and clusters of papers based on frequently occurring themes, using a systematic and objective procedure. Second, SMSs help plan new research, avoiding effort duplication. Third, they identify areas suitable for future systematic literature reviews (SLRs), a more in-depth form of secondary studies with a focus on smaller research areas and more concrete research questions compared to SMSs. The software engineering community is working towards the definition of a standard processes for conducting SMSs. Guidelines and procedures for undertaking SMSs are defined in Budgen et al. (2008); Petersen et al. (2008a, 2015). Similar to other studies (e.g., da Mota Silveira Neto et al., 2011 and Tofan et al., 2014), we split the process proposed by Petersen's (Petersen et al., 2008b) into three main phases (see Fig. 3.1):

- *planning the review*, where the need for the review, appraisal of related literature surveys and research questions are set. Similar to other SMSs (da Mota Silveira Neto et al., 2011; Tofan et al., 2014), we complement Petersen's. approach with a protocol definition process and the data collection form as suggested by Kitchenham and Charters (2007),
- *study identification*, where relevant papers are identified. First, a set of initial papers are identified by querying digital databases. Then, these studies are filtered based on inclusion/exclusion criteria, yielding primary studies.
- *data extraction and classification*, where primary studies are analyzed to derive the classification schema, and studies are classified under the schema.

Next subsections provide the details.

3.1. Phase 1: planning the review

This section introduces the directives for planning our SMS, along Kitchenham's guidelines (Kitchenham and Charters, 2007). This step iterates along three activities: *protocol definition, literature survey* and *research question definition* (see Fig. 3.1-“Phase1”). We analyzed literature surveys on SPL evolution whose outcome is presented in Section 2. As for the research questions, we point readers to the introduction, where the objective of research ques-

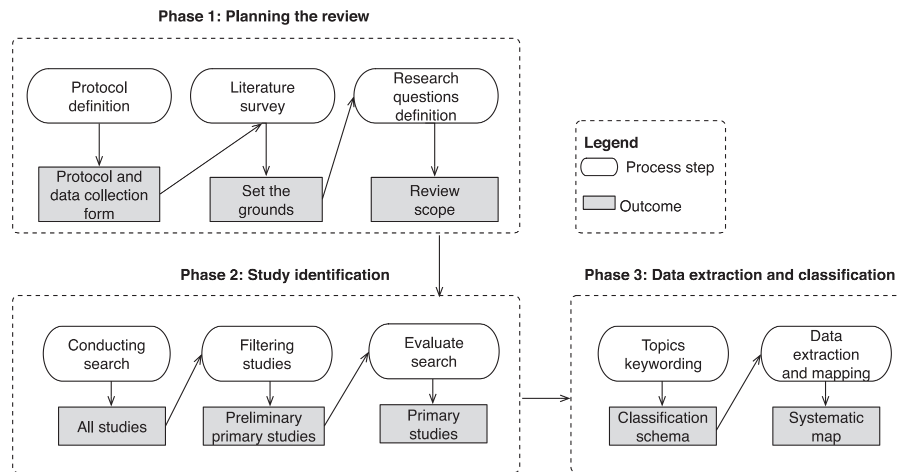

Fig. 3.1. Systematic Mapping Study process (adapted from (Petersen et al., 2008a)).

tions RQ1, RQ2, RQ3 and RQ4 is set. Hence, this section focuses on the protocol definition.

3.1.1. Protocol definition

This includes the need, the topic and the scope of the review, the preliminary research questions, a preliminary search strategy, selection criteria, and a data extraction form (Kitchenham and Charters, 2007). We reviewed and updated the protocol in several iterations throughout the entire SMS process.

The need for the review. This SMS is motivated by the perceived need to systematically map out efforts made on SPL evolution. Thus, the outcomes of this study can identify the trends, hotspots and gaps which need attention from the community. Moreover, leading venues to publish results (and read literature) on SPL evolution can be identified. In addition, researchers and practitioners can check if there is a growing or decreasing interest on SPL evolution. An overview of the field and its distinctive concerns is given at the beginning of this work (Sections 1 and 2).

Preliminary research questions. The goal of this study was to obtain a comprehensive overview of current research on SPL evolution.

The search strategy. The search strategy must lead to inclusion of relevant papers and exclusion of irrelevant papers. We set initial search strategy to include querying digital databases with customized search strings, followed by manual filtering of the resulting studies by predefined inclusion and exclusion criteria. To avoid replication, we detail this process later in Section 3.2.

Inclusion and exclusion criteria. For filtering the papers, we formulated inclusion and exclusion criteria. The inclusion criteria are:

- IC1. The study focuses on SPLs as opposed to peripherally addressing the topic.
- IC2. The study focuses on SPL evolution as such. Migration from single product to an SPL approach, refactoring, bad-smells and bug-fixing are not considered (as addressed in Section 2).
- IC3. The study is peer-reviewed.

Next, the exclusion criteria are:

- EC1. The study is not SPL-centric.
- EC2. The study does not address evolution.
- EC3. The study is in a language other than English.
- EC4. The study is gray literature, extended abstract, tutorial, tool demo, or doctoral symposium paper.
- EC5. The study is a delta of another study in the review.

Data extraction form. Its main purpose is to help researchers in collecting all the information needed to answer the research questions, recording rationales for inclusion and exclusion of the studies, and classifying each of the studies along the classification schema. We employed a spreadsheet to collect metadata for all of the studies: *title, authors, year of publication, publication type, venue, abstract*, and *keywords*. Additionally, we gave a brief *summary* for each study and *rationales* for inclusion or exclusion. If a study was included, then we determined its classification *categories*. The resulting table for all primary studies is available at http://www.onekin.org/content/spl-evolution-mapping.

3.2. Phase 2: study identification

This phase includes: *conducting the search* and *filtering studies*. Additionally, we added the *evaluating the search* step to verify that we did not miss any important study (see Fig. 3.1-“Phase2”). Fig. 3.2 depicts the process.

3.2.1. Conducting the search

This step deals with building a search string to query digital databases. We followed the PICO approach as suggested by good practices on systematic reviews (Petersen et al., 2015; Kitchenham and Charters, 2007). *P* stands for *population*. In our case, population refers to the area on *SPLs. I* stands for *intervention*. In our case, the procedure to be assessed is *evolution. C* corresponds to *Comparison*. Here, we do not compare different strategies for evolution but assess the area as a whole. Finally, *O* stands for *Outcome* which does not apply to our study either. The identified keywords are then, “Software Product Lines” and “Evolution”.

Next, synonyms should be found. Along the guidelines of Petersen’s (Petersen et al., 2015), the following related mapping studies were consulted: (Khurum and Gorschek, 2009; Chen and Babar, 2011; do Carmo Machado et al., 2014; Laguna and Crespo, 2013). Additionally, we conducted a pilot study over the IEEE database to find a balance between *hits* and *noise*. We noticed that the terms “evolution” and “maintenance” tend to be used interchangeably. Hence, we included both terms. This resulted in the following search string:

((“product lines” OR “product families” OR “product family” OR “product-lines” OR “product-families” OR “product-family”)
AND
(”evolution” OR “evolving” OR “maintenance” OR “maintaining”))

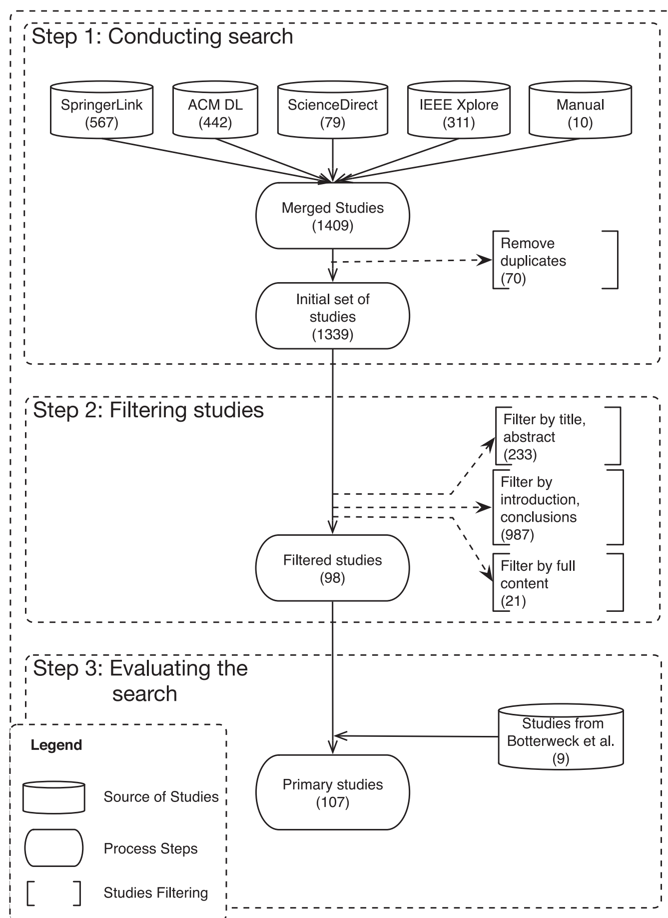

Fig. 3.2. Study identification process.

We restricted the search to studies published up to **July 2015**. The following electronic databases were consulted: IEEE Xplore,[3] ACM Digital Library,[4] Springer Link[5] and Science Direct.[6] The query was matched against the *title*, the *abstract* and the *keywords*. Unfortunately, at the time of this study, Springer did not account for such focused search, and we resorted to posing the query against the full article content. Additionally, previously known references (identified during the analysis of related literature in the "planning" phase) were manually added. Refer to Fig. 3.2 to inspect the number of the studies that each digital database returned. Fig. 3.2-"Step1" highlights how Springer Link returned most primary studies (40,2%). Next, ACM Digital Library, Science Direct, IEEE Xplore, and manually retrieved studies, returned 31,4%, 5,6%, 22,1% and 0,7%, respectively. In summary, we obtained 1409 primary studies in this first step, where 70 were duplicated and hence, removed. This leads to 1339 initial studies.

3.2.2. Filtering studies

For filtering, we formulated inclusion and exclusion criteria (already presented in Section 3.1.1). A paper was selected as a primary study only when it met all the inclusion criteria and none of the exclusion criteria. Filtering was mainly conducted by one researcher. When the researcher was not sure about including or excluding a paper, the other researcher was asked to discuss and decide. Next, we outline the main debates:

- EC1 ("The study is not centric to SPL"). Some studies addressed SPLs incidentally, not really focusing on SPLs. For instance, studies just mentioning SPLs as related work (e.g., Ahn and Chong, 2007).
- EC2 ("The study does not address evolution"). We found that *evolution* might encompass a great variety of concerns such as migration or refactoring. As noted in Section 2, we understand *evolution* as "activities to make enhancements". Hence, we left outside activities such as SPL migration (Breivold et al., 2008; Laguna and Crespo, 2013), SPL bad-smell detection (Abdelmoez et al., 2004; Loesch and Ploedereder, 2007; Devine et al., 2014; Bertran et al., 2010; Padilha et al., 2014; Vale et al., 2014) or SPL refactoring (Alves et al., 2008, 2006; Ribeiro and Borba, 2008; Schulze et al., 2012, 2013) or SPL bug-fixing (Krishnan et al., 2011, 2013). We also excluded studies on traceability with a focus on trace extraction and trace specification (Anquetil et al., 2010; Merschen et al., 2012; Moon et al., 2007; Ahn and Chong, 2007; Shen et al., 2009; Vianna et al., 2012; Yu et al., 2012).
- EC4 ("The study is grey literature"). We excluded grey literature, and also extended abstracts, tutorials, tool demos, and doctoral symposium papers (e.g., Vierhauser et al., 2014; Dhungana et al., 2007; Czarnecki, 2007).
- EC5 ("The study is a delta of another study in the review"). 26 deltas were excluded in favor of the paper that more extensively detailed the issue (e.g., ter Beek et al., 2011; Botterweck et al., 2009; Weyns et al., 2011).

We applied a three-stage filtering process to the initial set of 1339 studies (see Fig. 3.2-"Step 2"). Filter 1 looks at the title and abstract (233 papers left out). Filter 2 looks at the introduction and conclusions (987 papers left out). Finally, filter 3 looks at the content (21 papers left out). At a given stage, a study was filtered out only if the researcher doing the work was fully sure that it met all the exclusion criteria and none of the inclusion criteria. Otherwise, it went to the next filtering stage. If reaching the third stage, the study was revised by the two researchers, and a consensus was reached. The process resulted in 98 primary studies.

3.2.3. Evaluating the search

The filtering of studies was mainly conducted by one researcher, which is a threat we were aware of. To reduce the risk of having missed any important study, we followed (Petersen et al., 2015) guidelines, which recommend to cross-check the resulting studies with a test-set of studies. Our test-set was extracted from the most up to date summary on SPL evolution by Botterweck and Pleuss (2014). From the set of Botterweck's references we excluded those that do not met our inclusion/exclusion criteria, and obtained a final test-set of 34 studies. We then cross-checked these 34 studies with our 98 primary studies. The cross-check revealed 9 new references. This rises the number of primary studies to 107.

3.3. Phase 3: data extraction and classification

This phase iterates along two tasks, *relevant topics keywording* and *data extraction and mapping* (see Fig. 3.1-"Phase 3").

3.3.1. Relevant topic keywording

This process yields the classification schema. Our classification schema includes four facets: "Research type", "Product-derivation approach", "Asset type" and "Evolution activity". The classification schema is grounded in the literature. Specifically, the "relevant topic keywording" process was performed to refine the categories for facet "Evolution activity". We departed from a coarse-grained classification for "Evolution activity" first proposed by Yau et al. (1978). This classification was refined by means of the "relevant

[3] http://ieeexplore.ieee.org.
[4] http://dl.acm.org/.
[5] http://link.springer.com/.
[6] http://www.sciencedirect.com/.

topic keywording" process. Within this process, a reviewer read the papers and manually look for keywords and concepts that reflected the contribution of the papers. Afterwards, the set of keywords from the different papers were combined together and clustered to form the fine-grained categories for the "Evolution activity" facet. The resulting fine-gained schema is later presented in Section 4, as part of the mapping of primary studies. Next paragraphs provide the description of the four facets.

Facet 1: Research type. Description & derivation method. The research type reflects the research approach used in the primary study. As other SMSs in software engineering (Engström and Runeson, 2011), research type categories are based on the scheme proposed by Wieringa et al. (2006).

Classification Schema:

- "Experience papers" describe the experience of the authors, usually in practice, using a certain method, technology, etc. Often, these papers are written by people from industry.
- "Conceptual proposals" sketch a new way of looking at existing things, providing a vision or philosophical view on a subject matter.
- "Solution proposals" describe a solution which is usually illustrated with an example, case study, running example, etc. The work is barely or not validated; the proposal is only explained, and it is shown how to apply it.
- "Validation research" describes validation of research that is not deployed in practice, for example, by an experiment, performing some kind of tests, lab studies, etc. Usually it follows a solution proposal. It answers the question: is the proposed solution "good"?
- "Evaluation research" describes an evaluation of research, usually by seeing how the solution works in practice or comparing it with other solutions, pointing out positive and negative points. It is more extensive than validation and often carried out within an industrial setting. It answers the question: is the proposed solution the "right" solution?

This facet somehow serves as an indication of maturity. For instance, the existence of case studies or prototype tools in an academic context indicates at least a certain degree of validation ("Solution proposals" and "Validation research"). On the other hand, "Experience papers" and "Conceptual proposals" might denote an incipient research area.

This classification schema is disjointed, i.e., a study belongs to a unique category. If a study addresses two categories (e.g., a solution and its validation), the "uppermost" category is selected (e.g., validation). From a maturity perspective, categories rank as follows: "Evaluation research" > "Validation research" > "Solution proposals" > "Conceptual proposals" > "Experience papers". Note that both "Validation research" and "Evaluation research" will cover studies that propose *new* solutions (if they are validated or evaluated), as well as papers that address the validation or evaluation of *existing* solutions. Hence, we could not determine whether solutions being evaluated/validated are new or they have already being proposed. For our purposes, this is not an issue since our emphasis is on determining the maturity level of each research area, regardless of whether solutions are new or not.

Facet 2: Product-derivation approach. Description & derivation method. It refers to the way products are obtained from core assets. Two approaches are commonly distinguished: annotation-based (a.k.a. negative variability) and composition-based (a.k.a. positive variability) Apel et al. (2013). However, if the abstraction level of assets is also considered, a number of studies also address model-driven SPLs. A minority yet practical approach for product derivation is the use of clone-and-own.

Classification Schema:

- "Annotation-based". Here, the code of all features is merged into a single code base, and annotations spot which code belongs to which feature. *During product derivation*, all code that belongs to deselected features is removed (at compile time) or ignored (at run time) to form the final product (Beuche et al., 2004; Krueger, 2001). Pre-processors are a case in point. They typically provide facilities for conditional compilation, where marked code fragments in the source code are conditionally removed at compile-time. Annotations are realized through tags, such as *#ifdef* and *#endif*.
- "Composition-based". Here, features are realized as compassable units, ideally one unit per feature. *During product derivation*, all units of all selected features and valid feature combinations are composed to form the final product. Frameworks (Johnson and Foote, 1988), Component-based development, Feature-Oriented Programming (FOP) (Batory et al., 2004; Prehofer, 1997), Aspect-Oriented Programming (AOP) (Kiczales et al., 1997) or Delta-Oriented Programming (DOP) (Schaefer et al., 2010) applied to SPLs fall within this category.
- "Model-driven". Here, code is abstracted in terms of models. During *product derivation*, model transformations are used that, ideally, generates the complete product together with all documentation, test cases, etc., in a fully automated way (Greenfield and Short, 2003; Völter and Visser, 2011). Model-driven SPLs can follow annotations or composition for variability handling. For our purpose, however, the distinctive aspect is that they abstract the way at which product derivation takes place, let this be "annotation-based" or "composition-based".
- "Clone-based". In early stages of SPL adoption, developers might prefer keeping clone-based generated products separately. Here, *product derivation* is just "clone-and-own". Nevertheless, those products conform a family, where changes in one product might need to be propagated directly to sibling products without the intermediation of an SPL infrastructure (Rubin et al., 2015).
- "Hybrid". This comprises studies that somehow combines or blend some of the aforementioned approaches.

This classification schema is disjointed, i.e., a study belongs to a unique category. Papers addressing model-driven SPLs are so classified, no matter whether annotation or composition is used. In this way, we want to gain a glimpse to the extent model transformation is being involved in product derivation.

Facet 3: Evolution activity. Description & derivation method. Activities involved in SPL evolution. We tap into the change mini-cycle model of Yau et al. (1978).

Classification Schema:

- "Identify change". Customers, product engineers, domain engineers, the target market, maintenance needs or competitors might exert evolutionary forces over an SPL. "Identify change" has to do with monitoring those sources of change.
- "Analyze and plan change". Program comprehension is essential to understand what parts of the software will be affected by a requested change. In addition, the extent or impact of the change needs to be assessed to obtain an estimation of how costly the change will be, as well as the potential risk involved in making the change. This analysis is then used to decide whether it is worth carrying out the change.
- "Implement change". This activity conducts the change. The large number of assets and stakeholders involved in SPLs recommend error prevention and guidance mechanism to be in place.

- "Verify change". Techniques to re-verify the SPL after change are crucial to ensure that the SPL integrity has not been compromised.

This classification schema allows for categories to overlap, i.e., a study might belong to more than one category. A finer-grained schema is later presented in Section 4, as part of the mapping of primary studies.

Facet 4: Asset type. Description & derivation method. Type of the SPL asset being subject to evolution. Types are obtained from the reviewed studies.

Classification Schema:

- "Variability model". Variability modeling is to efficiently describe more than one variant of a system. Different approaches to capture such variability have been proposed: Feature Models (FMs) (Kang, 1990), cardinality-based FMs (Kim and Czarnecki, 2005), Decision-Oriented Variability Models (DOVMs) (Schmid et al., 2011), and Orthogonal Variability Models (OVMs) (Pohl et al., 2005).
- "SPL architecture". An SPL architecture captures the structure commonalities and structure variability of the SPL products, along the architecture elements: software assets, the externally visible properties of those assets, and the relationships among them (Capilla et al., 2014).
- "Code assets". Broadly, code assets are the raw material to produce the SPL products. This can range from code snippets to models (in model-driven SPLs). Here, code asset might enclose variability built-in, later resolved during product derivation.
- "Products". Broadly, a product is what is delivered to a customer. Depending on the maturity of the SPL, products might be directly derived from the reusable assets based on feature selection, or rather, require the intervention of product engineers before being ready for release (Deelstra et al., 2005).

This classification schema is "overlapped", i.e., a study might address evolution for different assets. Notice however, that studies are classified based on the "evolving artefact", i.e., the artefact that suffers the change first, regardless of whether this change is next propagated to other artefacts. So, a study describing how a change in the variability model percolates to code assets and products, is classified as "Variability model".

3.3.2. Data extraction and mapping

Having the classification scheme in place, the primary studies are sorted into the scheme. The classification scheme evolved while doing the data extraction, like adding new categories or merging and splitting existing categories. Data extraction process was performed by one reviewer, who entered data into the data extraction form fields: (i) gave a short description of each paper's contribution, (ii) classified the study into the four facets, and (iii) provided a short rationale why the paper should be in a certain category. The second reviewer checked the outcome of this process and checked its correctness. The outcome of this second review could be *agreement, disagreement* or *doubt*. If *disagreement*, the document was read (again) in full appraisal by both researchers, and a consensus was reached. If the classification was still dubious, then the studies' authors were contacted through e-mail. This was the case for 15 papers. Additionally, we contacted authors of other 13 studies, as a cross-check measure. These 28 studies are listed in the acknowledgements to thank the authors for the prompt reply to our request. The mapping of the papers and their brief is provided in Section 4. The Appendix holds Table A.1 with the mapping of the primary studies into our classification schema.

3.4. Threats to validity

There are several factors that may threaten the validity of systematic mapping outcomes. Main shortcomings include: (i) bias in the selection of studies (Barney et al., 2012), and (ii) errors when extracting and classifying studies into detailed categories (Engström and Runeson, 2011). Additionally, we evaluate this mapping study along Petersen's evaluation rubric (Petersen et al., 2015).

3.4.1. Selection of studies

Biases might happen during both finding and filtering primary studies. The former has to do with coming up with primary studies. Here, one of the risks is the lack of standard languages and terminologies (Dybå and Dingsøyr, 2008). To reduce this risk, we refined the "search string" by (i) consulting the keywords used on related mapping studies, and (ii) conducting a pilot study, which let us determine the "noise" introduced by the selected keywords. Additionally, we referred to the main publishing houses in computing science (i.e., ACM, IEEE, Springer and Science Direct), even knowing that a large overlap could exist (indeed, 70 duplicates were detected). Inclusion and exclusion criteria were established to provide an assessment of how the final set of primary studies was obtained. Where in doubt, the screening of a study went from the abstract, introduction and conclusions, to the full-text appraisal. If after full text appraisal, doubts persisted, then the decision about whether to include or not the study was jointly taken by the two researchers. This was the case for 21 primary studies (see Fig. 3.2).

In addition, we follow recommendations by Casteleyn et al. (2014) to set aside "delta papers", i.e., papers that provide minor additions compared to previously published work of the authors. Inclusion of delta papers might mislead summarization data, specifically if classification is fine-grained with few studies for each facet. This process led to the identification of 26 delta papers. As a final validation, we conducted a cross-check with the two main potentially overlapping survey studies, i.e., (Laguna and Crespo, 2013; Botterweck and Pleuss, 2014). Specifically, primary studies of Laguna and Crespo (2013) present no overlap with our primary studies. As for Botterweck and Pleuss (2014), though this work is not a mapping study but a survey, their references serve to cross-check ours: 25 overlapping, 9 only in Botterweck, and 73 only in our study. Besides enriching our set with 9 new references, this comparison corroborates the role of our work as a systematic mapping endeavor by introducing 73 new references.

We cannot rule out threats from a *quality* assessment perspective because selected studies were assigned no scores.[7] However, with the aim of increasing the quality of included studies, we defined exclusion criteria to get rid of potentially low level quality studies, such as those excluded by "EC4" (grey literature, extended abstract, tool demo, workshop proposal). Additionally, the selected digital databases (ScienceDirect, ACM, IEEE Xplore, and SpringerLink) which are regarded as reliable by the community. Some systematic reviews that include them are: Dybå and Dingsøyr (2008); Laguna and Crespo (2013); do Carmo Machado et al. (2014).

Another threat might be the focus on those studies that specifically target SPLs. We did not explore whether other software engineering studies addressing evolution could be applicable for SPLs. Moreover, our notion of *evolution* can be regarded as too restrictive as we did not consider SPL migration or SPL refactoring. As for the reviewers' reliability, the authors of this study are researchers in SPLs with three papers in the SPLC conference, being (Montalvillo and Díaz, 2015) the one more related to SPL evolution.

[7] In SMSs, *quality assessment* is not a mandatory practice (Petersen et al., 2008a).

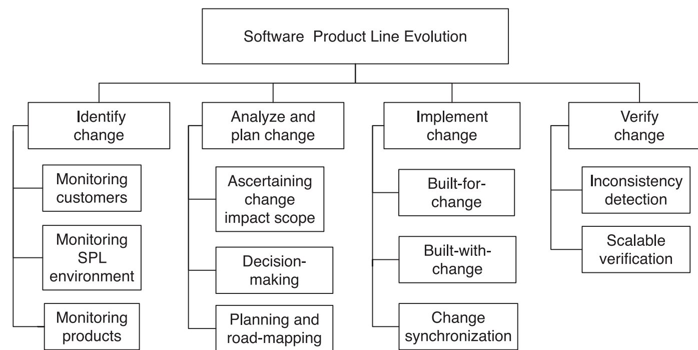

Fig. 4.1. Elaborating on the "Evolution activity" facet.

3.4.2. Classification errors

It is possible for authors to introduce bias during the data extraction process. To reduce this risk, we based the data extraction on the words used in each publication wherever possible. First, an author conducted the data extraction and classification process. The outcome of this second review could be *agreement, disagreement* or *doubt*. If disagreement, the document was read in full (full appraisal), and a consensus was reached. If the classification was still dubious, then the document's authors were contacted through e-mail. This was the case of 15 papers. As a crosscheck, we additionally contacted authors of 25 papers, although only 13 did finally reply. No inconsistencies were appreciated except for the facet "Research type": 5 authors would classify their paper differently w.r.t to this facet. The main confusion originated from the distinction between "validation" and "evaluation" research. Additionally, some authors misunderstood when a study should be considered an "experience paper". This is not totally unexpected. Wohlin et al. (2013) already pointed out how misleading this facet can be. The authors reveal how two independent studies classified the very same papers differently, w.r.t the "Research type" facet. This blurriness might advise to stick to the classification of a single observer that makes clear his understanding of this facet's values, and where the assessment of which research type was conducted is based uniquely on what it is described in the paper. The alternative would be to collect the answers of the 67 studies' authors whose understanding of what "validation" and "evaluation" is might differ, and whose appreciation might be partially biased from experiences not always fully documented.

3.4.3. Evaluation rubric for this mapping study

Petersen et al. (2015) devise an evaluation rubric where to assess the quality of a mapping study process. This rubric can be used for readers to quick assess the actions undertaken in a SMS. Specifically, authors identify 26 actions worth applying. The more actions taken, the higher would be the quality of a SMS. Table 2 outlines the actions undertaken in this SMS. Additionally, we include a fourth column, which points to the Section in which the action is addressed. According to the findings of Petersen et al., the median quality of the analyzed SMSs is 33%. This SMS undertakes 15 out of the 26 suggested actions, which yields a ratio of 57%.

4. Mapping of primary studies

This section provides a short summary for the primary studies. This implied a more carefully reading not just of the abstract but the whole content. This permitted a finer-grained elaboration of the facet "Evolution activity" based on the challenged addressed by the primary studies (see Fig. 4.1). Table A.1 provides the outcome. Next, we dedicate a subsection to each of these nine activities. For each activity, we first outline what makes this activity challenging for SPLs. Next, we provide a brief about how these challenges are addressed in the primary studies.

4.1. Identify change

SPLs broader scope and larger life-span make asset evolution unavoidable. Asset evolution happens in response to forces both outside the SPL organization and within it. By monitoring these forces, engineers can identify emerging needs that the SPL may support. Studies differ in the *force* being monitored: customers, SPL environment, or products (i.e., product engineers).

4.1.1. Monitoring customers

Customer needs can be identified through requirement volatility analyses. Requirement volatility is the tendency of requirements to change over time in response to evolving needs (Peng et al., 2011). In SPLs, requirement volatility tends to be higher due to its broader scope. Here, requirement volatility analysis helps to predict which requirements might change and how. The analysis is based on the priorities that customers assign to each of the SPL requirements. Hence, by monitoring changes to these priorities, engineers identify the set of the requested adaptations, e.g., new requirements may arise, others become obsolete, others may shift from mandatory to optional, etc. This approach is investigated by Savolainen and Kuusela (2001) and Villela et al. (2010). An SPL requirement-based taxonomy is provided by Schmid and Eichelberger (2007).

4.1.2. Monitoring the SPL environment

Discussion forums, competitors' websites and market studies can provide useful data silos where to mine future SPL needs. Böckle (2005) discusses measures to monitor the SPL environment, including: (1) workshops and discussion forums, (2) usability labs where customers can play with new products and where ideas and complaints are collected, (3) prototypes for new products, and (4) competitors.

4.1.3. Monitoring products

Product engineers are responsible for providing feedback to domain engineers. To spur product-engineer feed-backing, Carbon et al. (2008) adapt the agile practice "planning game" Planning Game to SPLs. By means of so-called *reuse stories*, product engineers are instructed to provide concrete suggestions about how to improve the reusability of SPL assets. In addition, product engineers might develop product-specific assets. These assets may "inspire" domain engineers. This is illustrated by Mende et al. (2008) and Creff et al. (2012) where code analysis tools are developed to identify product-specific assets candidate to be promoted as SPL core assets.

Table 2
Actions conducted in this SMS: taken (✓) & not taken (•).

Phase	Actions	Applied	Refer to ...
Phase 1	Motivate the need and relevance	✓	Introduction & Background & Protocol definition (Sections 1 & 2 & 3.1)
	Define objectives and questions	✓	Introduction & Protocol definition (Section 3.1)
	Consult with target audience to define questions	•	–
Phase 2	**Choosing search strategy**		
	Snowballing	•	–
	Manual	✓	References from (Botterweck and Pleuss, 2014) (Section 3.2.2)
	Conduct database search	✓	ACM, IEEE, SpringerLink & ScienceDirect (Section 3.2.1)
	PICO	✓	Phase 2: data collection (Section 3.2.1)
	Consult librarians	•	–
	Iteratively try finding more relevant papers	✓	Conduct a pilot study (Section 3.2.1)
	Keywords from knows papers	✓	From papers (Khurum and Gorschek, 2009; do Carmo Machado et al., 2014; Chen and Babar, 2011; Laguna and Crespo, 2013) (Section 3.2.1)
	Use standards, encyclopedias, and thesaurus		–
	Evaluate the search		
	Test–set of known papers	✓	Test-set references from (Botterweck and Pleuss, 2014) (Section 3.2.2)
	Expert evaluates result	•	–
	Search web-pages of key authors	•	–
	Test–retest	•	–
	Inclusion and Exclusion		
	Identify objective criteria for decision	✓	Inclusion and exclusion criteria (Section 3.1)
	Add additional reviewer, resolve disagreements between them when needed		–
	Decision rules (what to do when doubts)	✓	Postpone paper to next filtering level & ask second reviewer (Section 3.2.2)
Phase 3	**Extraction process**		
	Identify objective criteria for decision	✓	Provided along the classification schema (Section 3.3.1)
	Obscuring information that could bias	•	–
	Add additional reviewer, resolve disagreements between them when needed	✓	We asked authors of 28 studies (Section 3.4.2)
	Test–retest	•	–
	Classification scheme		
	Research type	✓	Facet "Research type" included (Section 3.3.1)
	Research method	•	–
	Venue type	✓	Venues and frequencies reported (Fig. 5.1)
Validity disc.	Validity discussion/limitations provided	✓	Validity evaluation reported (Section 3.4)

4.2. Analyze and plan change

Even to a larger extent than for single products, SPL assets exhibit numerous dependencies: (1) intra-feature dependencies (e.g., *<excludes>* or *<includes>* dependencies in variability models); (2) feature-to-code dependencies (a.k.a. configuration knowledge) or (3), product-to-feature dependencies, which are tracked through product configurations. This coupling makes changes rarely be a one-off event. Hence, *ascertaining the change impact scope* is a first step to decide whether or not to carry out the change. This requires of *decision-making* processes tuned to the kind of change being considered. For instance, changing the variability model does not have the same implications than changing a code asset. If the change goes ahead, then *planning and road mapping* come into play. Next, we look into these issues.

4.2.1. Ascertaining the change impact scope

Change Impact Analysis (CIA) is defined as "identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change" (Bohner, 1996). CIA scope depends on the asset at hand. Variability models are those with broader impact when evolved. This explains why CIA for variability models has received most attention. But it is by no means the only one. Table 3 depicts different change scenarios arranged along the root of the change ("source") and its ripple effects ("target"). Note that it is possible for a study to give support to more than one scenario. Next we provide a paragraph for each row.

A change in the variability model might impact .... the variability model itself. Paskevicius et al. (2012) resort to Prolog rules to assess how changes in the Feature Model (FM) affect other parts of the FM. The FM is expressed in Prolog. For instance, the rule *fm :- all(alt(f1), f2, f3)* describes a FM with *f2, f3* as compulsory features, and *f1* as optional. FM changes are also captured as Prolog rules. When the FM is changed, the rule engine computes the set of features affected by the change as a result of the existing feature dependencies (e.g., excludes, includes, and feature associations). The output is the set of features impacted by a change. Heider et al. (2012c) present an industrial case study, where they identify engineers' desired trace links when performing CIA in a component-based SPL. Desired traces include links between the variability model and solution space assets (e.g., components, interfaces, and dependencies between them) to ascertain how changes in the variability model impacts the solution space. They further discuss implications for a tool support CIA based on the eclipse IDE.

... code assets. Livengood (2011) describes industrial experience on assessing CIA for large and complex variability models (those having multiple constraints). Specifically, authors stress how difficult it is to determine how implementation is affected when variability model constraints are modified. So far, the organization relies on engineers to determine the impact of such changes. Authors advocate for enhanced traceability between the variability model and the code assets.

... product configurations. Changes to the variability model may alter the configuration space (e.g., introducing a new feature adds new product configurations). Thüm et al. (2009) present an algorithm to reason about the impact of FM changes on product configurations. The algorithm takes the two versions of the FM (i.e., before and after the change) and classifies changes as follows: (1) *generalization*, if the set of valid product configurations is extended with additional alternatives, (2) *refactoring*, if the same configurations exist, (3) *specialization*, if the set of valid configurations is

Table 3
CIA scenarios.

Triggering Source\Triggered Target	Variability model	Architecture	Code asset	Product
Variability model	Paskevicius et al. (2012), Heider et al. (2012c)	Heider et al. (2012c)	Livengood (2011), Heider et al. (2012c)	Thüm et al. (2009), Dintzner et al. (2015), Murashkin et al. (2013), Heider et al. (2012b), Michalik and Weyns (2011), Heider et al. (2012c),
Architecture	Heider et al. (2012c)	Heider et al. (2012c), Díaz et al. (2014)	Heider et al. (2012c)	Michalik and Weyns (2011), Heider et al. (2012c)
Code asset	Heider et al. (2012c)	Heider et al. (2012c)	Yazdanshenas and Moonen (2012), Jiang et al. (2008), Pichler et al. (2011), Heider et al. (2012c), Ribeiro et al. (2014)	Heider et al. (2012c), Michalik and Weyns (2011)
Product	–	–	Käßmeyer et al. (2015)	Rubin et al. (2013, 2012)

reduced, and (4) *arbitrary* change, if some of product configurations are removed and others are added. Similar goal but for multi SPLs (i.e., a set of interacting and interdependent SPLs) is presented by Dintzner et al. (2015). Murashkin et al. (2013) develop a visual tool to detect the set of product configurations that become non-optimal when the FM changes (w.r.t. quality attributes). In their approach, FMs are annotated with quality values, e.g., *cost* and *usability*. Product configurations are also annotated with expected objectives, e.g., product configuration *p1* can have at most a *cost* of 1500, and *usability* must range between 100 and 300. When a feature quality value evolves (e.g., the cost of a feature increases), the tool highlights those product configurations that do not fulfill the set objectives.

... already derived products. Changes in the variability model may force products to be updated accordingly. Michalik and Weyns (2011) propose a preliminary CIA model where to keep track of derived products' configurations, so that whenever the FM changes engineers can assess the affected products. Heider et al. (2012b) introduce a tool for domain engineers to get feedback on how changes performed to the variability model may affect existing products. Given a new version of the variability model, the tool re-generates existing products according to their configurations. Next, the tool triggers regression tests for domain engineers to assess the impact of these changes on the re-generated products.

A change in the SPL architecture might impact .... the SPL architecture itself. Architectures are the result of design decisions. If those decisions are recorded and contextualized through the features, then so-captured design decision can help to trace core components back to features. This is the insight of Díaz et al. (2014). Consider an ATM SPL. Let's *balanceAccount* be a feature about providing information about user account balance. This feature provides context for the design decision: *"if there is an overload of requests, reject it"*. This decision is in turn traced back to the architecture component that implements it (e.g., *Balance* component). On changing feature *balanceAccount* (e.g., adding new variations or excluding dependencies), CIA can go down to the potentially affected components. This scenario gets more complex when design decisions might rest on other design decisions so that their algorithm goes down until all affected components are ascertained. Authors evaluate their approach in an industrial case study on smart grids.

... already derived products. Changes in either the component dependencies or the bindings between these components and the features, may force products to adjust to the new arrangement. Michalik and Weyns (2011) propose a preliminary CIA model where to keep track of derived products' configurations. Heider et al. (2012c) present an industrial case study, where they identify engineers' desired trace links when performing CIA. Desired CIA include assessing SPL architecture changes on (1) derived products, (2) dependencies with other architectural components and interfaces, and (3) features in the variability model.

A change in code assets might impact .... the variability model. Changes to component interfaces and component dependencies often affect variability models (Heider et al., 2012c). Heider et al. (2012c) present an industrial case study. They identify engineers' desired CIA, including how code assets changes affect variability models. A model is generated based on those desires and a possible realization in Eclipse is discussed.

... code assets themselves. *Clone&own* is not limited to products. Code assets can also be obtained by cloning existing code assets. In this setting, Jiang et al. (2008) present an automated technique to identify code asset that need to be changed when a code asset changes. For component-based SPLs, Yazdanshenas and Moonen (2012) introduce a fine-grained source code analysis (at code line level) where the impact of component line-grained changes in other components is assessed. For annotation-based SPLs, Ribeiro et al. (2014) develop an Eclipse-based tool for annotation-based SPLs. Given a point in code (the one to be changed), this tool identifies the set of additional code changes associated to other features that need to be addressed for the change to be completed. For model-driven SPLs, Pichler et al. (2011) and Corrêa et al. (2011) tackle change impact on meta-models and model transformations. Pichler et al. (2011) envisage ten changing scenarios and their respective scopes are analyzed. For instance, a change into a meta-model might ripple through the meta-model itself, model-to-model transformations or model-to-text transformations. Based on the classification for meta-model changes proposed by Gruschko (2007), Corrêa et al. (2011) adapt it for model-driven SPLs. For instance, *Non-breaking changes* (NBC) in SPLs are those changes that do not break consistency and variability rules, and therefore, no product is affected. Authors classify changes in model-driven SPLs (feature changes, meta-model changes and transformation changes) according to this classification, and identify eventual ripple effects.

... already derived products. New enhancements in reusable code assets might impact already derived products. Michalik and Weyns (2011) proses a preliminary CIA model that keeps track of the configuration details for each derived products.

A change in a product might impact .... code assets. Improvement opportunities can be detected by product engineers. Käßmeyer et al. (2015) tackle this scenario. For Version Control Systems (VCSs), development histories can be used to trace products back to the SPL release version from where the product was initially derived. Previous release versions that hold the targeted asset can be

Table 4
Classification of studies based on the decision to be taken.

Decisions to be made	Primary studies
Make Variable/Make Common	Thurimella and Bruegge (2007), Thurimella et al. (2008), Thurimella and Bruegge (2012), Deelstra et al. (2009), Noor et al. (2008), Riva and Rosso (2003), Liu et al. (2007), Annosi et al. (2012), Schackmann and Lichter (2006), Sarang and Sanglikar (2007), Peng et al. (2011), Chen et al. (2004)
Make Generic/Make Specific	Heider et al. (2010b)
Product-local change	Gámez and Fuentes (2013), Gámez and Fuentes (2011), Karimpour and Ruhe (2013)
New product	Chen et al. (2004), Heider et al. (2010a), Tran and Massacci (2014), Murthy et al. (1994), Schmid and Verlage (2002)

detected as well, which, in turn, permits to identify which other SPL products might benefit.

... already derived products. In clone-based SPLs, changes made to one clone might be propagated to other clones. Rubin et al. (2012) propose a model to describe information for managing cloned products. Herein, if a clone changes, then this model could point to other affected features within the clone as well as identify other impacted cloned products. The authors discuss the realization through VCSs. In a later study (Rubin et al., 2013), authors approach is validated through a set of industrial case studies.

4.2.2. Decision-making

A change request is not a must-do. Developers should first explore the impact of conducting a change. This very much depends on the kind of change being conducted. This subsection classifies studies based on our understanding of the change type being considered. Change types are those indicated in Fig. 2.1. Table 4 pigeonholes studies based on these change types. Note that it is possible for a study to give support for more than one change type.

Make variable/make common. Here, the issue is about finding the right amount of variability. Too much commonality moves the SPL towards traditional single product engineering. On the other hand, more variability broadens the SPL scope at the expense of more maintenance (and upfront investment). On the search for a compromise, decision-making approaches come in handy, specifically, the *WinWin* model (Boehm et al., 1994) and the *Question Options Criteria* (QOC) model (MacLean et al., 1991).[8] Thurimella and Bruegge (2007) propose a combination of the *EasyWinWin* model (i.e., an adapted version of the *WinWin* model) and the *QOC* model. Specifically, the model includes a *question* (e.g., *"what are the changes that have been requested for feature F1?"*), a set of *options* (e.g., changing variability from mandatory-to-optional, from optional-to-mandatory, to add a new feature or to delete a feature), and finally, some *criteria* (e.g., cost to implement each of these options). In this way, Thurimella and Bruegge (2007) adapt *QOC* to SPLs. Alternatively, Thurimella et al. (2008) and Thurimella and Bruegge (2012) enrich variability models with annotations about feature rationales. This information can later be used to assess *what* and *how* to manage variability. This approach is later evaluated by Thurimella and Brügge (2013).

In the same vein, Deelstra et al. (2009) introduce the variability assessment method COSVAM. COSVAM requires engineers to provide both (1) the SPL's variability model, and (2), the required variability (i.e., the variability necessary to accommodate the change request). The tool detects mismatches between the provided and the required variability. If mismatches arise (i.e., existing product configurations become invalid), the tool suggests the set of adaptations needed to overcome the mismatches. However, estimating the cost of such changes is not always easy. Predictive modeling is a process used in predictive analytics to create a statistical model of future behavior. Schackmann and Lichter (2006) and Sarang and Sanglikar (2007) advocate to create such models from past evolution-driven developments efforts. These models can later be used to estimate costs for the different SPL evolution scenarios (e.g., fix a feature, add a new feature, etc.). At this respect, Peng et al. (2011) assess the profit that a change would imply. The metric is based on the following estimates: (1) the probability that the change will emerge (estimated by analyzing the market and the technological trends), (2) the volume of the change (the number of products affected by the change) and (3), the added customer value for each product (estimated by multiplying the price and the relative value of all the impacted problems identified in change impact analysis).

If the focus is on risks assessment, Riva and Rosso (2003) present an industrial case study, where architectural assessment helped to determine if a new feature would put under risk the SPL integrity. Architectural assessments are used to identify defects and shortcomings of the SPL architecture. If the architecture is weak, new features may compromise the integrity of the SPL. Here, SPL managers may postpone the new feature until the architecture is ready to support it. On the other side, new features may alter the functionality of already existing features. Hence, a careful analysis of feature interactions is vital. Liu et al. (2007) focus on the identification and modeling of safety-critical feature interactions to determine whether they may cause a hazard. For component-based SPLs, Annosi et al. (2012) present an industrial experience on risk management when updating COTSs.[9] The upgrade may surface incompatibilities with other features resulting into unforeseen side effects. Authors build a decision model that considers expert knowledge and dependencies between the SPL architecture elements (i.e., existing components) and the COTS candidates.

Make generic / make specific. Here, the issue is about making a variable asset product-specific ("make specific") or a product-specific asset generic ("make generic"). For this matter, Heider et al. (2010b) resort to a *WinWin* model. Key stakeholder roles are first identified (e.g., salesperson, product engineers, customers, SPL managers), and next, negotiation clusters are set (e.g., development, market, management). For each negotiation cluster, stakeholders describe their individual objectives and expectations as *win* conditions. For instance, project managers might favor cheap and fast development while product engineers prefer to develop with reuse despite introducing additional delays. If all stakeholders concur on a *win* condition, then the condition is turned into an *agreement*. Otherwise, stakeholders identify conflicts, risks, or uncertainties as *issues*. Stakeholders seek *options* to overcome the collected issues and explore tradeoffs as a team. *Options* can then be turned into *agreements* that capture mutually satisfactory solutions.

[8] *QOC* models arrange decision making along four steps. First, define the issues (*questions*). Second, identify available solutions (*options*). Third, define the criteria (e.g., estimates about development efforts, benefits and risks) to rate the available options. Finally, a decision (*option*) is selected on this basis.

[9] COTS are pre-packaged solutions usually acquired to a third-party for a fee.

Product-local change. When core assets are enlarged with a "newcomer", a question arises about which SPL products to be used as a test bed. Karimpour and Ruhe (2013) tackle this issue. They compute the synergy between the newcomer and distinct products in terms of *value* and *integrity*. The *value* is provided by products' customers, based on how much value will be added to the product if the newcomer is incorporated. The *integrity* computes cohesion, i.e., the degree to which (a product's) features are perceived to be related to the newcomer (e.g., *play* and *pause* features of a video-player systems are more cohesive than *play* and *volume* features). The best product candidate would be the one with maximum *value* and *integrity*. In a similar vein, but now focusing on product architectures, Gámez and Fuentes (2013) resort to *diff* tools to compute the architectural differences between a product's current configuration and the new configuration that will emerge, should the newcomer be incorporated. The output identifies which components must be added or removed from each product. Managers would then assess the cost for producing the upgraded product versions.

New product. SPLs can potentially account for a large number of products based on different feature combinations. However, not all products end up being realized. The cost of a product is not limited to generating the product. Besides the potential pressure for product-local changes, a new product is a new asset to be maintained when the SPL evolves. This begs the question: how to decide the introduction of a new SPL product? Studies resort to simulation models. Simulations involve designing a model of a system and carrying out experiments on it as it progresses through time. Here, the model is the SPL ecosystem, and the experiments are about the impact of introducing the new product. Studies differ in the estimate being considered, e.g., development effort, time-to-market, change resiliency or marketability.

Chen et al. (2004) resort to simulations to estimate the development effort and time-to-market. SPL managers should first create the model, indicating: the number of current SPL products, phases on which the different products are (development, release, waiting for core assets requested), phases on which core assets are (in development, released), and the number of developers and their current state (free or under development activities). SPL managers can next simulate the desired change (e.g., introducing a new product). The simulation will tell managers about: the time-to-market for the new product, its development effort, and the additional maintenance effort caused by the change. Effort estimates are traditionally obtained based on previous development efforts. Alternatively, simulation of evolution scenarios can be used. For model-based SPLs, Heider et al. (2010a) resort to this approach to measure model maintenance effort.

Tran and Massacci (2014) aim to predict products' resiliency. Experts specify the prediction of future evolutions in a feature-like model (called *eFM*). Based on both the *eFM* and the current feature model, authors provide a *configuration survivability analysis* for new product configurations. This analysis measures whether a configuration would still be operational in the presence of forthcoming evolutions.

Murthy et al. (1994) tackle marketability. *Product marketability metrics* are proposed to capture customer affordability (willingness to pay) and product quality. Though the study focuses on single applications, the authors argue that these metrics can also be useful to assess whether a new product should enter an SPL. An interesting issue is whether product introduction is a one-off event or rather, it might be better to introduce several products as a single shot. Schmid and Verlage (2002) discuss the economical impact of these two scenarios.

4.2.3. Planning and road-mapping

The change backlog rarely holds a single petition. Rather, distinct changes are often competing for attention and resources. Harmonious evolution requires roadmaps and release plans that guide the evolution journey.

Road-mapping. A project roadmap is a simple presentation of project ambitions and project goals alongside a timeline. The aim is to manage stakeholder expectations, and generate a shared understanding across the teams involved. For SPL evolution, a roadmap provides a global vision of the SPL with features and products to be offered some years from now. Pleuss et al. (2012) and Schubanz et al. (2013) propose the use of FMs to describe roadmaps. Such FMs are called EvoFM, which include "the what" and "the why" of the change. EvoFMs are composed of FM fragments. A fragment gathers related features that are added or removed together during the same evolution step. Dependencies between fragments can also be established, just like in an standard FM. Each evolution step can then be represented by a "configuration" of the EvoFM, i.e., a selection of fragments that together make a FM. The evolution of a FM can, hence, be represented by a sequence of EvoFM configurations. Authors visualize this sequence in a matrix-like roadmap. The horizontal dimension represents the time line (year), where each column represents an evolution step. Each cell in the plan represents a configuration decision, i.e., whether a FM fragment is applied in that version or not. Moving from FMs to SPL architectures, van Ommering (2001) proposes for SPL roadmaps to include both products and components, and most importantly, release dependencies between them. Finally, Savolainen and Kuusela (2008) report experiences from industrial SPLs and suggests key factors for effectively road-mapping, including e.g., decomposing features into sub-features (to better understand feature inter-dependencies), mapping features to component versions (to understand how features are mapped to code), and prioritizing features based on the value that each product gives to each feature.

Release planning. A release plan is a company's current understanding of what features are going into the next release, how many effective developers are deployed on it, and the current status of the development effort (ahead, behind, on-time). It differs from road-mapping in that it signifies that there are a subset of selected requirements to be implement, and there are committed resources to implement such requirements. Release planning provides focus to road-mapping. To know which requirements should be part of the next release, requirements prioritization is conducted. Prioritization can be based on distinct criteria: costs and benefits (Noor et al., 2008), constraints on available resources to conduct the requirements (e.g., person months until next release) or dependencies between requirements (e.g., one requirement includes/excludes another) (Inoki et al., 2014). The large set of concerns to be considered leads (Taborda, 2004) to specify release plans as matrixes with different layers. Each layer accounts for different SPL release facets: prioritized product features, allocated requirements for each component, estimated development effort, scheduled dates, test plans cases, and delivered product configuration. The author describes the results of practical trials.

4.3. Implement change

CIA strives to identifying the potential consequences of a change. The aim is collecting data to decide whether the change ends up being implemented or not. If the answer is yes, then we move to "Implement change". For classification purposes, we arrange studies addressing this activity along three main issues: (1) how to make SPL assets change resilient ("Built-for-change"),

(2) how to accommodate change in a reliable way ("Built-with-change"), and (3), how to ensure consistency when changes are scattered across different assets ("Change synchronization").

4.3.1. Built-for-change

Studies strive to anticipate change, and reflect about means to make assets change resilient (Loughran and Rashid, 2004). Resilience very much depends on the SPL architecture and the programming paradigm used to implement code assets.

SPL architecture resilience. Studies strive to make the SPL architecture steady through evolution. For planned changes, the wired-in variability of SPL architectures accommodates well. However, unplanned changes might compromise the SPL architecture stability. The question is how to ensure long-term viability of SPL architectures considering that unplanned changes are unavoidable. Although no golden-rules exist, Tischer et al. (2012) and Dikel et al. (1997) present some successful industrial cases. In hindsight, authors propose some guidelines: focusing on simplification (finding a balance between features that are needed for "tomorrow" and features that are needed for "today"), adapting for the future (forecasting market and technology trends that are specific to the SPL architecture), establishing architectural rhythm (fix regular architecture and product releases that help coordinate the actions and expectations of all parties), partnering and broadening relations with stakeholders (e.g., when users want changes to a component, they should negotiate directly with the component owner rather than directly change it themselves), maintaining a clear SPL architecture vision across the company (all parties need to know who is responsible for what), and managing risks and opportunities (e.g., review the architecture with customers and stakeholders, tracking and testing the assumptions underlying customer requirements). Deng et al. (2005) discuss several evolution challenges for SPL architectures, and propose a model-driven approach based on automated domain model transformations. Authors advocate that their approach is flexible enough to accommodate changes to the SPL architecture. Finally, Díaz et al. (2014) propose an SPL architecting approach that combines (1) an incremental SPL architecture development based on *scrum sprints*, and (2) a modeling technique to specify the SPL architecture and design decisions that led to each architectural element. Authors evaluate whether their approach enables to maintain SPL architectures' flexibility and integrity upon evolving requirements.

Code asset resilience. A number of studies evaluate how variation mechanisms perform as for change resilience. Traditional programming paradigms have been assessed by Svahnberg and Bosch (2000) and Sharp (1999). Svahnberg and Bosch (2000) compare inheritance, extensions, parameterization, configuration and generation. Additionally, Sharp (1999) discusses object-oriented mechanisms, including inheritance, aggregation, generic programming, and conditional compilation. Departing from traditional programming paradigms, newer approaches have been investigated for SPL realization, namely, Aspect-Oriented Programming (AOP), Feature-Oriented Programming (FOP), and Delta-Oriented Programming (DOP).

AOP supports cross-cuts, i.e., functionality that cannot be cleanly decomposed and tangles/scatters around distinct assets. Tesanovic (2007) endorse AOP as a suitable paradigm to face cross-cutting evolution. Dyer et al. (2013) compare different AOP interface proposals, namely, open modules, annotation-based pointcuts, explicit join points and quantified-typed events. Figueiredo et al. (2008) evaluate AOP strengths and weaknesses compared to conditional compilation in a set of evolution scenarios. Finally, Abdelmoez et al. (2012) contrast the maintainability effort required during evolution of aspect-oriented SPLs and object-oriented SPLs.

Next, FOP, i.e., a composition-based approach that provides the notion of feature as a construct of the programming language. The idea is to decompose code in terms of features (i.e., feature modules). Object-Oriented Programming (OOP) resorts to subclassing for extending a class C1 with additional functionality in subclass C2. In the same scenario, FOP defines a single class C1 but its definition is split into two assets: the *base* and the *feature* so that C1 is obtained by composing *base* • *feature*. There are not two classes but a single class that is incrementally extended to exhibit a new feature. Ferreira et al. (2014) evaluate FOP in several evolution scenarios. Authors conclude that FOP seems to be more effective tackling modularity degeneration, by avoiding feature tangling and scattering in source code, than conditional compilation and design patterns. Cafeo et al. (2012) compare AOP, FOP and conditional compilation. Cardone and Lin (2001) propose *java-layers* (JL), a FOP-like approach for Java, and evaluate JL against Object-Oriented frameworks in terms of flexibility, ease of use, and support for evolution.

Finally, DOP. DOP generalizes FOP by allowing removal of functionality, and hence, brings non-monotonicity to SPLs. In DOP engineers start from a *core module* (containing a valid product configuration), and apply *deltas* to remove, add, and modify features. Schaefer et al. (2010) introduce DOP, and compares it w.r.t. FOP in an SPL evolution scenario.

From the previous studies, it can be concluded that there is not a *one-size-fits-all* approach. Hence, hybrid approaches are suggested. *Aspectual feature modules*, a mix between AOP and FOP, is proposed by Gaia et al. (2014). Similarly, Loughran and Rashid (2004) evaluate *framed aspects*, a mix between AOP and *frames technology* (i.e., a language independent textual pre-processor that creates software modules by using code templates and a specification from the developer). Finally, for component-based SPLs, Tizzei et al. (2011) propose *aspectual-components*, a mix between AOP and components. Authors evaluate to what extent this approach supports the evolution of SPLs compared to object-oriented SPLs.

4.3.2. Built-with-change

SPL complexity substantiates the efforts to bring assistance during change implementation. Studies differ in the asset being the subject of change.

Changing the variability model. Error prevention can be ameliorated through constraints to be obeyed when conducting the change. Romero et al. (2013) follow this approach by allowing domain engineers to define authorized changes to the SPL. Such authorized changes are specified in a model (the evolution model). This model is next fed to asset editors so that editions should be compliant with the evolution model (i.e., the constraints). Similarly, Borba et al. (2012) and Teixeira et al. (2015) propose the use of templates. Templates regulate the SPL evolution so that the behavior of the original SPL products is preserved.

Changing the SPL architecture. Guidance to conduct change at architectural level is addressed by Hendrickson and van der Hoek (2007); Knodel et al. (2006) and Garg et al. (2003). The first two resort to a *diff-like* approach to capture differences between the architecture *as-is* and the architecture *as-it-needs-to-be*. This representation states the architectural elements (components, interfaces and connectors) that need to be added, deleted or modified. This assists engineers in determining the changes to be made. Similarly, Garg et al. (2003) present a tool to visualize different versions of architectural models in terms of components and connectors. When a change is implemented at code level, architecture evaluations can then be used to compare the architectural model with its corresponding implementation at code level. This assists developers in determining whether the changes have been thoroughly completed.

Changing code assets. Introducing changes at code level can be error-prone. This is more so for composition-based SPLs where code tends to be scattered across a large number of modules. For example, a module can reference classes, variables, or methods that are defined in another module. Safe composition guarantees that a product synthesized from a composition of modules is type-safe. While it is possible to check individual products by building and then compiling them, this does not scale. In an SPL, there can be thousands of products. It is more desirable to ensure that all legal modules are type-safe without enumerating the entire product line and compiling each product (Delaware et al., 2009). Schröter et al. (2014) introduce a tool for FOP, which tells engineers (while developing), whether their development is type safe, and hence, no compilation errors will await when composed with other modules. For AOP SPLs, Menkyna and Vranic (2009) advocate to create a *change catalog*. Once the type of change is identified (e.g., *Adding Column to Grid*), this catalogue helps to get an idea of its realization through AOP constructs (e.g., *Performing Action After Event*). Authors present this catalog using a Web application as a case study. Finally, Ribeiro et al. (2014) address the ripple effect among code assets in annotation-based SPLs. Based on usage dependencies between code snippets (e.g., variables, methods), a tool highlights the impact that changes in the definition of either variables or method signatures, have on other snippets using these elements.

Changing products. Customers might request product-specific changes. Product engineers might proceed by developing the bespoken code from scratch. However, the SPL mindset recommends to tap into the available SPL's code assets to look for re-use opportunities. Kakarontzas et al. (2008) assist product engineers on this matter by selecting the component that offer better reuse opportunities. Using Test-Driven Development, product engineers might resort to SPL components' test cases for both developing and testing the bespoken code.

For model-driven SPLs, code assets are realized in terms of models, and products are obtained through model transformation. Therefore, product specifics should be handled at the model level. But this is not always possible, and product-specifics end up being added at the code level. The issue is that once models become out of sync, any future re-generation of code overrides manual modifications. To solve this problem, Jarzabek and Trung (2011) propose a flexible model-to-text generator. The idea is to let engineers weave arbitrary manual modifications into the generation process rather than directly modify the generated code.

4.3.3. Change synchronization

Change synchronization looks at ways to restore consistency. For classification sake, we distinguish between "inconsistency detection" (addressed in Section 4.4.1) and "change synchronization" (this subsection). The former checks whether SPL assets are kept in a consistent state. The answer is basically "yes" or "no". On the other hand, "change synchronization" takes a step further by restoring consistency. Studies propose restore actions for different SPL assets. Differences stem from the asset being restored.

Scenario: keeping the variability model in sync. The variability model can hold a set of dependencies/constraints among its features. It is not enough to detect that some of these dependencies no longer hold. The triggering change should be followed by restore actions such as deleting a feature's children, or removing a cross-tree constraint. Guo et al. (2012) introduce a tool to assess those actions for cardinality-based feature models. Dhungana et al. (2010) introduce a tool to propagate changes between fragments of Decision-Oriented variability models.

However, keeping the variability model in sync is not limited to the variability model itself. It might also impact product configurations, which were set in terms of the variability model, which is now being updated. Unlike the previous case, now restore actions are not taken at the time the variability model changes but rather, it is up to product engineers to decide when it is the right moment for products to be upgraded. This decoupling requires of a *variability model change log.* This log records *who* made *what* at *when* to the variability model. Based on this log, product engineers can adapt product configurations at the time that they consider most appropriate. Heider et al. (2012a) tap on this log to assist product engineers in setting some constraints to be followed when working out the new product release w.r.t. the upgraded variability model. Gámez and Fuentes (2013) consider this scenario for cardinality-based FMs. Constraint-compliant configurations are obtained which might include new features in order to meet the constraints (e.g., to satisfy a *require* dependency). Barreiros and Moreira (2014) face large FMs where the options to restore product configurations might be very large. Authors introduce an algorithm based on the distance between the original configuration and a potential repaired configuration akin to the upgraded FM. The algorithm suggests those with the minimum distance. Finally, Kim and Czarnecki (2005) also tackle change propagation but for staged configurations.[10]

The variability model might also be impacted by changes conducted down in the SPL infrastructure. In the automotive domain, Holdschick (2012) consider how potential changes in the so-called functional model (e.g., deletion of components, optional component becomes mandatory) need to be propagated up to the variability model (e.g., reformulate relations with related features, split features, etc.).

Scenario: keeping architectures in sync. Usually, product architectures are first derived from the SPL architecture. From then on, both the SPL architecture and the product architectures might evolve independently. Domain engineers can extend the SPL scope and upgrade the SPL architecture accordingly. Likewise, product engineers might be forced to make changes to products architectures to ensure accurate and responsive customer service (Clements and Northrop, 2001). Temporary deviations between the SPL and product architectures are allowed, but periodic synchronizations might need to be performed. Notice that the triggering change might come from either the domain realm or the product realm.

If the change originates in the SPL architecture (i.e., the domain realm), then products might benefit from including the new enhancements in the next product release. To know how a product architecture should be updated, architectural traces (i.e., those that trace elements from the SPL architecture to products) become vital to determine what to merge. Michalik et al. (2011) seek to abstract the level at which this process is conducted. Although SPLs tend to describe their architecture through a model, this is not always the case for products where the architecture might be hidden within the code assets. This leads Michalik et al. to follow a modernization-like approach where the product's architecture model is first obtained from the product's code; next, this model is enhanced from the improvements conducted in the SPL architecture model; and finally, the so-obtained enhanced model is mapped back to code. The enhancement stage is conducted by comparing the current product's model and the SPL architecture model. These differences will lead product engineers to manually update products.

If the change originates in the product architecture, domain engineers might consider the change of interest for the entire SPL organization. Again, this process is decoupled, i.e., domain engineers

[10] Staged configuration is a process whereby product configurations are arrived at in stages. At each stage some feature choices are made.

do not consider product changes at the time the change happens, but at a later time in accordance with their roadmap. This begs the question of how engineers cherry-pick the interesting changes from the distinct ones the product suffers from the last milestone. Chen et al. (2003) tap into the product's version. First, domain engineers look at the product's version. Second, two versions are selected that isolate the change of interest. Second, differences are obtained. Third, these differences are accommodated into the SPL architecture through an *ad-hoc* algorithm. Unfortunately, the interesting change does not always correspond to one of the product's version. It might well be the case that interesting changes are scattered across different versions. This might substantiate the effort of Shen et al. (2010) to detect interesting changes from the differences between the current product architecture (no matter the number of releases it has suffered) and the current SPL architecture. Once differences are worked out, domain engineers pick those of interest, and merge then back to the SPL realm.

Scenario: keeping code assets in sync. Previous scenario looks at synchronizing architecture assets. Now, we tackle a similar scenario but for code assets. The difference stems from synchronization to be achieved not just between assets but asset versions. Versions introduce variability in time: the very same asset might be available along different versions. This means that products are derived from asset versions, not just assets. The very same core asset might be included in different products but at different stages of its life-cycle (i.e., different version numbers). Hence, versioning becomes a main synchronization factor. This moves us to VCSs. VCSs are designed to keep track of who did what and when. Broadly, VCSs support "revisions", i.e., a line of development (a.k.a *baseline* or *trunk*) with branches off of this. The branching model defines the strategy for branching off, and merging back (Walrad and Strom, 2002). Studies differ in the kind of product derivation process being addressed: *clone-based* and *composition-based*.

For *clone-based*, each product has its own repository. Several authors argue about the benefits of an integrated platform where cloned variants could be managed. Specifically, both Rubin et al. (2012) and Antkiewicz et al. (2014) propose conceptual operations to manage the synchronization of clones. An industrial experience on managing clone-based SPLs is later conducted by Rubin et al. (2013). Authors conclude that an efficient management of clones relies on not only improving the maintenance of existing clones, but also refactoring clones into an SPL infrastructure. From a technical perspective, McVoy (2015) introduces new VCS operations suited for BitKeeper, which enables opportunistic reuse and synchronization at component-level. Notice that in clone-based SPLs, propagation takes place at the level of products in the absence of a proper SPL infrastructure.

By contrast, *composition-based* SPLs derive products out of core assets. In a VCS setting, the SPL comprises: one SPL repository where to keep core assets, and distinct product repositories where to keep single products. Product repositories are derived from SPL repositories. A *link* between both repositories makes change propagations possible. Thao et al. (2008) present a home-made VCS tuned for component-based SPLs. Here, *special* branches inside the SPL repository, keep the SPL repository connected with product repositories. Whenever a product repository is derived, a *special* branch is automatically created in the core asset repository, aimed for change propagation. Specifically, the *special* branch references the product repository's trunk. This branch works like a mirror: if domain engineers merge changes from the SPL repository main development *trunk* to the *special* branch, the product repository will automatically get these updates. For Git/Github, Montalvillo and Díaz (2015) introduce a branching model geared towards facilitating the identification of un-synched assets and their respective change propagation. Here, *links* between the SPL repository and product repositories are created by means of *fork* links. Authors approach tuned operations provided by Github (*fork* and *pull requests*) for SPL specifics. When the SPL repository evolves, product engineers can automatically identify those parts in their product repositories that are no longer in sync, w.r.t. the code assets from where the product was derived. The user can assess the impact on those changes in a *diff* like manner, and enact propagation, if desired. Anastasopoulos (2009) and Dhaliwal et al. (2012) differ from the previous studies in keeping both SPL assets and product assets in the very same repository. For Anastasopoulos (2009), the vision is realized for the Subversion VCS. Engineers can perform activities related to evolution such as creating change requests for a given core asset, knowing if product assets are in sync with core assets' latest versions, and propagating changes between core assets and products. *Diff* operations are used to highlight the differences between core components and product components so that differences can later be merged into a product. However, integrating changes from the core-asset branch into product branches is not always easy. When the core-asset branch holds commits related to more than one change request (e.g., adding a new feature, updating an existing one, etc.), developers need to selectively cherry-pick the commits related to the change to be integrated. Commonly, change-request tracking systems (e.g., Jira) are used to keep the links between change requests and commits (e.g., a new feature *f* is implemented in commits *c1, c2* and *c3*). This way, product engineers select the change request they want to integrate into their products, and all the commits related to the change request are merged into the product branch. However, developers need to perform these tasks manually. This is error-prone and time-consuming. Dhaliwal et al. (2012) provide algorithms to identify commit dependencies and create groups of dependent commits that should be integrated together. Authors propose algorithms to automatically determine dependencies among the commits by analyzing dependencies among change requests (in Jira), structural and logical dependencies among source code elements, and the history of developers' working collaborations (in Git).

Scenario: keeping feature mappings in sync. Change propagation frequently requires a trace infrastructure to ascertain impacted assets. This infrastructure should also be upgraded. To this end, Seidl et al. (2012) introduce *re-tracing operations*, e.g., if *class C* is deleted, so should it be feature mappings that contain *class C*, provided domain engineers approve it. When feature traces are not specified into a separate asset but are embedded into code (i.e., feature annotations), Ji et al. (2015) present nine patterns for co-evolving code assets together with their embedded annotations. Finally, Passos et al. (2013) inspect the Linux kernel evolution history over four years to identify twelve patterns. These patterns cover how variability changes affect both feature-to-code mappings (specified through Makefiles) and source code embedded variability annotations (C files with annotated *ifdef* clauses). For instance, if a new optional feature is added, the pattern instructs engineers to add variability annotations into the source code, as well as to extend Makefiles to include the new feature definition.

4.4. Verify change

Once changes are conducted, the SPL needs to be revalidated to ensure that the SPL integrity has not been compromised (e.g., through regression testing). The issues are the specifics brought by the SPL assets and scalability. Rather than repeating all tests for each new release (should this be of the feature model, a core asset or a product), authors strive to find ways that scale to large SPLs to verify that the changes did not have inadvertent effects.

This subsection aligns with the mapping study on consistency checking presented by Santos et al. (2015). The results are quite

similar, though here we include a detailed description of the studies that is missing in Santos' et al.

4.4.1. Inconsistency detection

Inconsistency detection checks whether SPL assets are kept in a consistent state. The answer is basically "yes" or "no". Studies differ in the asset being checked.

Inconsistency detection for the variability model. Quinton et al. (2014) address consistency for cardinality-based feature models. Authors discuss about common changes and the resulting inconsistencies. A tool supports designers in assessing *the where, the why* and *the what* of the inconsistency. For decision-oriented variability models, Vierhauser et al. (2012) build a consistency checking framework where developers are given feedback about the constraints being violated at runtime (between the variability model and the code). However, changes in the feature model percolate down to the SPL, and hence, consistency checking should be extended to other assets, specially, product configurations. For instance, promoting a feature from optional to mandatory turns those configurations that do not included the upgraded feature, inconsistent. Besides product configurations, feature traces (i.e., those that link features to their code realization) are also likely to be affected. Consider a product configuration *p1* with features *f* and *g,* being class *F* and class *G* their code realization, respectively. Now, *f* is extended with an optional child (e.g., feature *h)* together with its corresponding code assets (e.g., class *H*). If class *H* is next inattentively mapped to feature *f* (rather than *h*), then product *p1* will no longer deliver the expected behavior. Borba et al., 2012 devise tools to check whether the behavior of already existing products configurations is preserved upon feature changes. In the same vein, Leopoldo Teixeira and Gheyi (2015) provide a theory about behavior preservation in SPLs upon feature changes. This study is later extended for multi-product lines (i.e., independently-developed SPLs that are later integrated) (Teixeira et al., 2015). Finally, Jahn et al. (2012) develop a consistency checker to detect inconsistencies for decision-oriented variability models w.r.t the SPL architecture model. When engineers change the code assets (e.g., new components are added), the SPL architecture is automatically updated. The tool raises warnings about any inconsistency between the variability model and the SPL architecture. The tool further suggest the engineer how to resolve such inconsistencies by proposing changes to the variability model (e.g., a new feature should be added).

Inconsistency detection for the SPL architecture. Different means are proposed in this item: regression testing, functional tests and architecture evaluations. Regression testing for SPL architectures, checks if new defects are introduced into a previously tested architecture. da Mota Silveira Neto et al. (2012) apply regression testing in two scenarios: corrective changes and perfective changes. Júnior and Coelho (2011) resort to JUnit tests to detect violations of design rules during SPL evolution. Alternatively, studies Knodel et al. (2006) and Duszynski et al. (2009) use architecture evaluations, i.e., the comparison of an architectural model with its source code counterpart. Possible outputs include: the architectural element converges (if it exists in both the architecture and the source code), the architectural element diverges (if it is only present in the source code) or the architectural element is absent (if the element is only present in the architecture). Next, architects can interpret the results based on the total numbers of convergences, divergences and absences. Finally, Knodel et al. (2006) illustrate how this output is used to evaluate the SPL architecture consistency between its design and the SPL code assets.

Inconsistency detection for products. When new releases for code assets are delivered, existing products might need to be accordingly upgraded. Due to frequent upgrades, products might keep unnecessary assets (a kind of bloatware). This superfluous code may be harmful in safety critical domains, hindering runtime performance and smooth evolution. Demuth et al. (2014) resort to functional tests for ascertaining and eliminating the *bloatware* assets from products, as well as for assuring consistency of products when code assets and variability model evolves.

4.4.2. Scalable verification

SPLs might include a large number of assets. Lowering verification efforts has to do with reducing the number of assets that need to be re-verified. Approaches differ based on the verification mechanisms being used: model checking, compositional reasoning and regression testing.

Model checkers automatically verify if a *system* satisfies a given *property*. A *property* can be concerned with safety or liveness of the program, such as the absence of deadlocks, but also product-specific behavior can be checked (e.g., in a coffee machine SPL, check that the total cost of a drink is always less than 2$). The *system* needs to be described in a formal notation (e.g., Petri nets, state-transition diagrams). For large SPLs, Cordy et al. (2012) resort to incremental verification. Here, previous verification results are used to minimize the re-verification effort. Specifically, authors try to determine if new added features are *conservative* or *regulative*. A feature is *conservative* to a product if it adds functionality to the product, without altering its previous functionalities. Alternatively, a feature is *regulative* if it doesn't add new functionality to the product but "adapts" previous functionalities. When the SPL evolves and a new feature *f* is implemented, knowing that *f* is *conservative* may drastically reduce the number of new products to verify. For instance, any property violated by an old product *p* is also violated by the new product *p* after *f* is added. Hence, if *p* is known not to satisfy a property, then there is no need to check *p* again. The scenario becomes more complex when a blend of both conservative and regulative features is added simultaneously. Theorems are provided to determine which subset of products can be left out for verification when such types of features are introduced. Similarly, static analysis techniques are used by Sabouri and Khosravi (2014) to determine which features *affect* which *properties* (a feature affects a property if it can make the property valid/invalid). In this way, when the SPL evolves (e.g., a feature is modified that adds/removes program statements), this technique identifies the affected *properties*. Here, there is no need to re-verify the properties that are not *affected* by the statement added/modified.

Berezin et al. (1997) introduce so-called "assume guarantee reasoning", a compositional model checking approach that verifies each component separately. It is based on decomposing the system specification into a set of *properties* each of which describes the behavior of a system's subset (i.e., components). Components are annotated through an *assume-guarantee* pair. *Assume* describes the properties for the correct functioning of the component. *Guarantee* denotes properties satisfied by the component provided the *assume* clause is met. A component's *assume* may depend on other component's *guarantee*. This approach is taken by ter Beek et al. (2012), where the SPL architecture is denoted as a set of components chained by *assume-guarantees*. When a component implementation changes, its *assume-guarantees* may change as well. If stable (the *assume-guarantee* pair did not change), products that reuse the component don't need to be tested again.

Similarly, Rumpe et al. (2015) resort to a component compatibility approach, based on pair-wise model checking. If a new component version is compatible with the previous version of the products' component, it could be safely replaced. Finally, common-

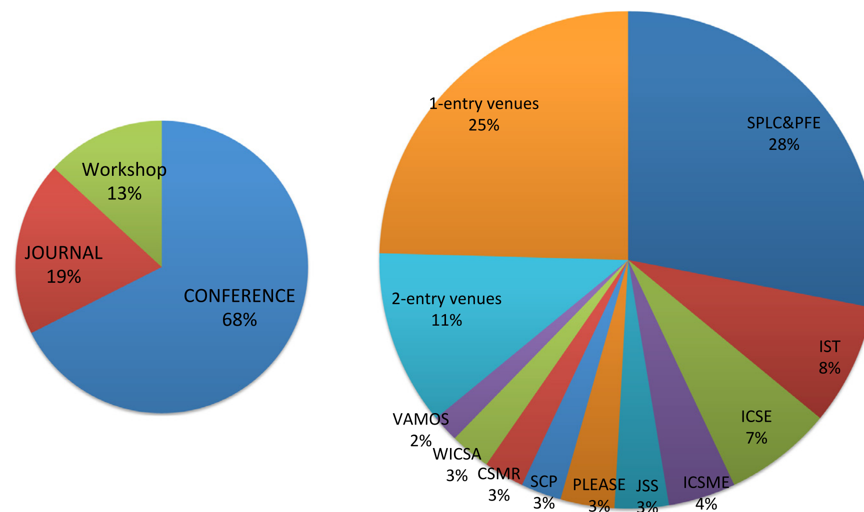

Fig. 5.1. Distribution of studies over publication venues: types (left) and individuals (right).

alities and similarities between products' configurations can be analyzed to additionally narrow the set of products to be tested. The idea is to determine a minimal set of products such that the successful verification of such a small set implies the correctness of the entire SPL. Scheidemann (2006) presents an algorithm for this matter.

Regression testing is a type of software testing used to determine whether new problems are the result of software changes (refer to Engström and Runeson (2010) for an survey for single product regression testing practices). The new twist brought by SPLs is that tests can also be core asset and hence, subject of reuse. For instance, Lity et al. (2012) use model-based testing in delta-oriented SPLs. When a new product is created, the commonalities with existing product configurations is ascertained, and test assets are automatically derived for the brand new product. In this way, product testing is given a head start.

5. Analysis of the results

Though it was not the main driver of this research, we depict distribution of studies over publication venues in Fig. 5.1. The International SPL Conference (SPLC) is the prime publication venue for SPL evolution research (28%). In 2005, the SPLC committee decided to merge the SPLC with its European counterpart, the Product Family Engineering (PFE) conference, so they are jointly visualized in the chart. Next in the ranking is the Journal on Information and Software Technology (IST) (8%), the International Conference on Software Engineering (ICSE) (7%), the International Conference on Software Maintenance and Evolution (ICSME) (%4), the Journal of Systems and Software (JSS) (3%), and the ICSE co-located International Workshop on Product Line Approaches in Software Engineering (PLEASE) (4%). The top ten is completed by the International Conference on Software Maintenance and Reengineering (CSMR) (3%), the Journal of Science of Computer Programming (SCP) (3%), the Working Conference on Software Architecture (WICSA) (3%) and the International Workshop on Variability Modeling of Software-Intensive Systems (VaMoS) (2%). The 25% of the publications were unique in the venue they were published in. Fig. 5.1 also depicts the type of publication venue. Conference papers and Journals account for the 68% and 19%, respectively, while workshops account for a 13%. These results align with the state of-the-art on SPL evolution by Botterweck and Pleuss (2014). Specifically, the majority of the included papers by Botterweck et al. belong to the SPLC (together with the ICSE). Additionally, we both agree on the low numbers of both the International Conference on Software Reuse (ICSR) and the Generative Programming: Concepts & Experience (GPCE) conference. Next, we address each of the research questions.

5.1. RQ1: What types of research have been reported, to what extent, and how is coverage evolving?

From the accumulated results shown in Fig. 5.2, we observe that "Solution proposals" (31%) is the most addressed category, followed by "Validation research" (24%). As it can be observed, "Solution proposals" have been gradually increasing over the years. "Evaluation research" accounts for a 19%, which indicate the maturity level of the SPL evolution field. Specifically, "Evaluation research" has been lately more increasingly conducted (from 2008 on). This might indicate the SPL field becoming more mature within an industrial setting. Additionally, "Validation research" (24%) studies conducted in academia still need to find their way to industry. "Experience research" (%17) indicates the commitment degree of industry to report "know how", "open issues" and "challenges behind". A few conceptual works have also been addressed (9%), which might indicate incipient challenges being addressed by the community.

From the stacked bar chart, we see a peak of contributions reached in 2012.[11] This peak aligns with other SPL related systematic reviews, which have also identified a global maximum in 2012 (Santos et al., 2015; Thüm et al., 2014). Santos et al. (2015) found also a global maximum with 7 studies (the 29%), while the rest of the years had less than the half of the studies found during 2012, except for 2010 (with 6 papers). Thüm et al. (2014) also identify a global maximum in 2012, with 27 papers. Regarding the evolution of the research, it comes as no surprise that during the first years (up to 2005) "Experience research" and "Solution proposals" are the ones most addressed. From then on, we observe a trend towards "Validation proposals" and "Evaluation research". Nevertheless, "Solution proposals" still are prominently addressed, which seems to indicate the existence of SPL evolution challenges left to be accomplished.

[11] Notice that the survey stops at July 2015. One could postulate that a similar number of papers could be published in the second semester of 2015.

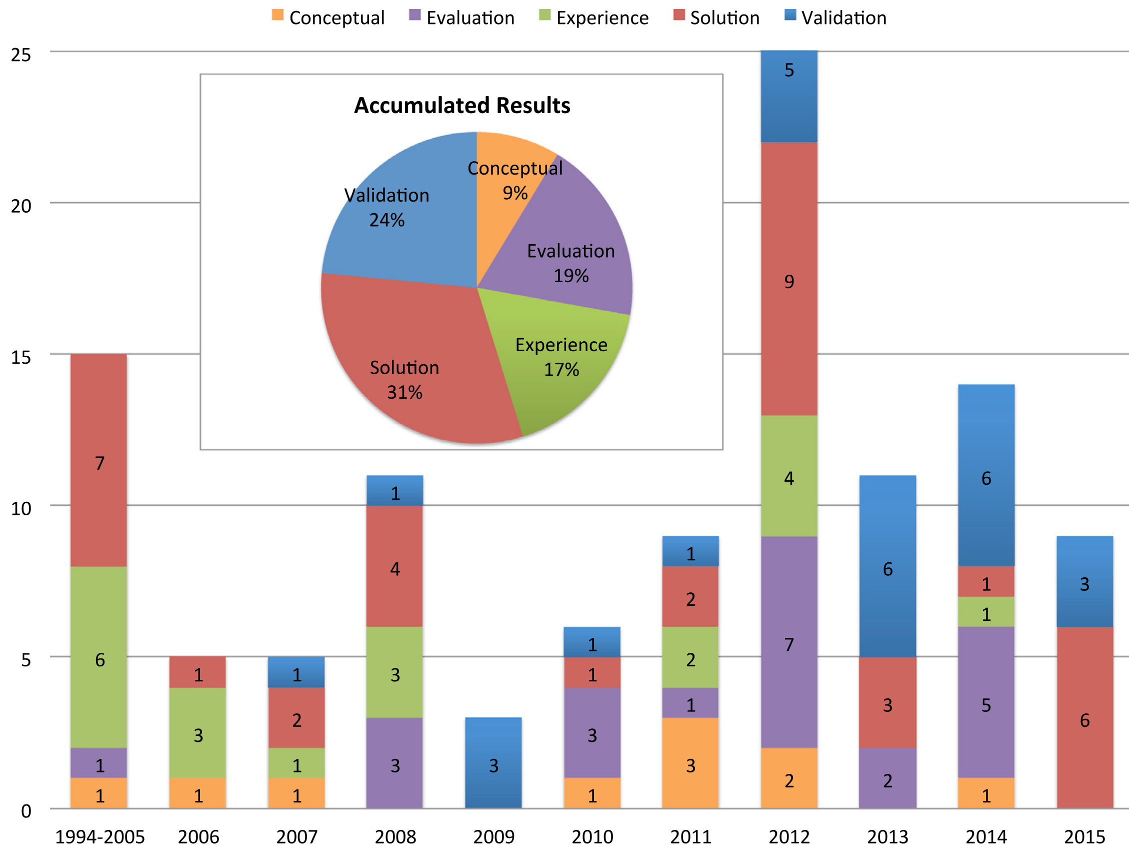

Fig. 5.2. "Research type" over time.

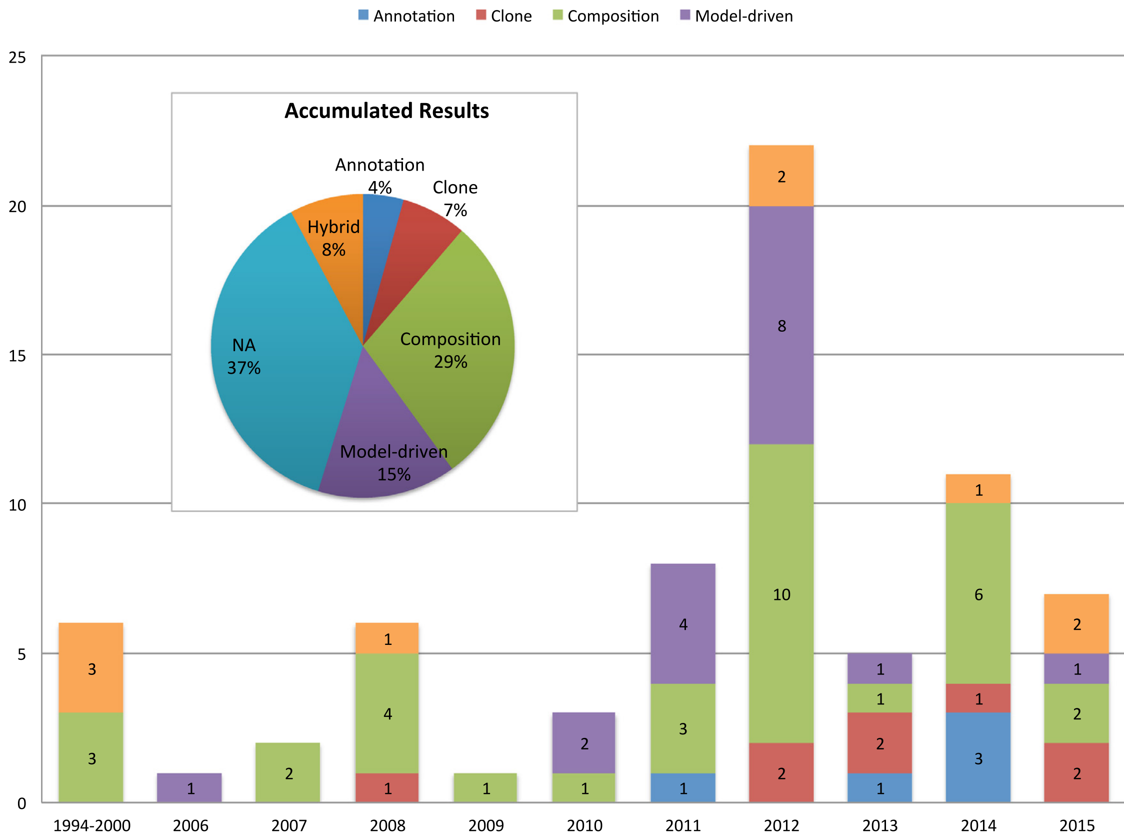

Fig. 5.3. "Product derivation approach" over time.

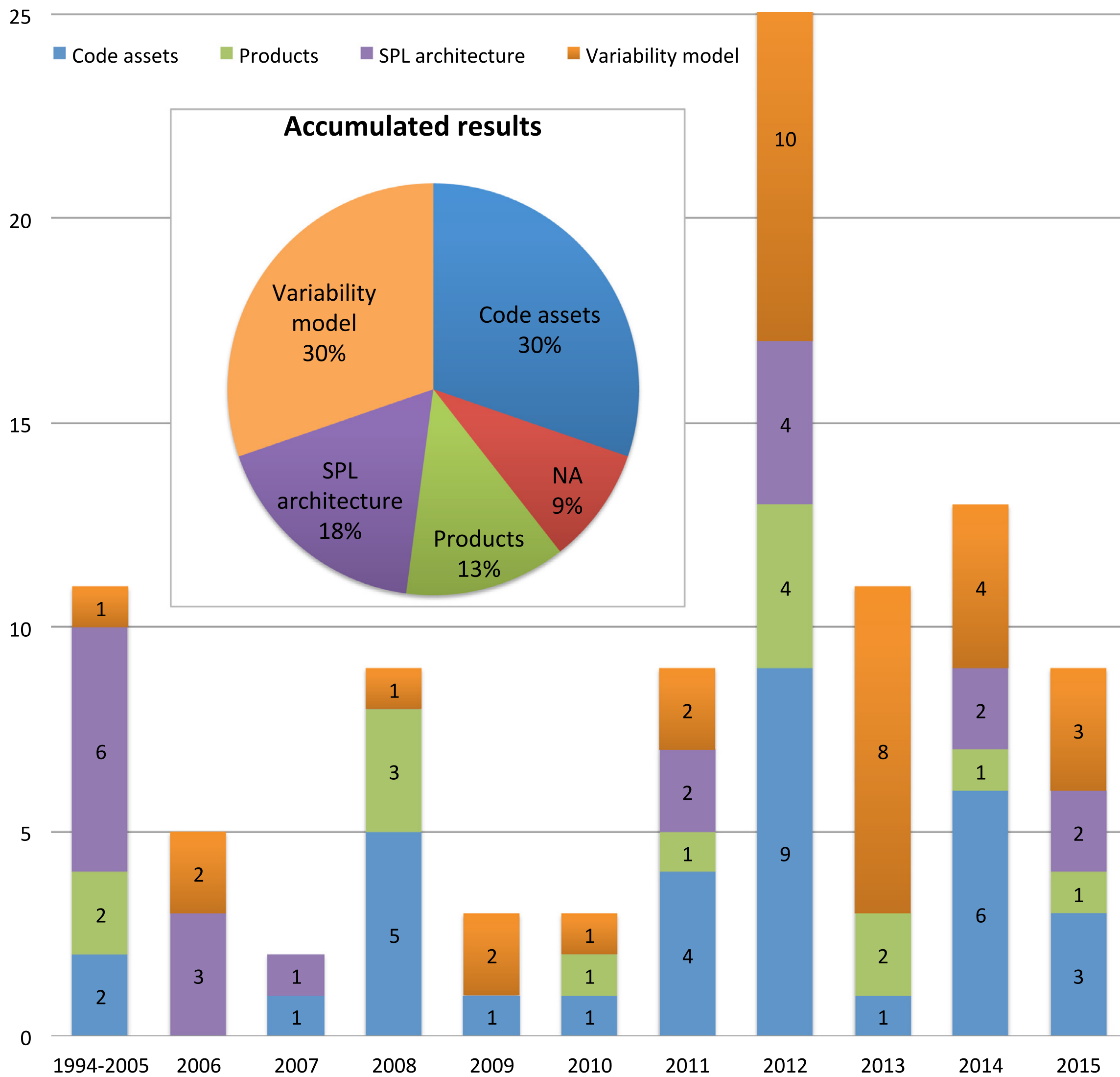

Fig. 5.4. "Asset type" over time.

5.2. RQ2: Which product-derivation approach received most coverage, and how is this coverage evolving?

We are interested in assessing how the distinct product derivation approaches are catching on (see Fig. 5.3). These approaches might, for instance, impact change implementation in so far as the structure and code assets might take different shapes tuned for the variability implementation and product derivation approach at hand. This in turn might affect how other activities are conducted from implementing to propagating change. The 37% of the studies are not reporting any specific product derivation approach.[12] The rest of the studies consider either "Annotation" (4%) (e.g., *#ifdef* clauses), "Composition" (e.g., component-based approaches, AOP) (29%), "Model-driven" (15%), "Clone" (7%) or "Hybrid" (8%) approaches. From the stacked bar chart, we can observe how the most addressed one is composition-based, with a share of 29%. This is at odds with the annotation approach being the most widely reported in industry (Ganesan et al., 2009; Jepsen and Beuche, 2009; Pearse and Oman, 1997; Tartler et al., 2009). This can be due to composition approaches being proposed to overcome the difficulties that annotation-based approaches face when evolved in the large (Ernst et al., 2002; Favre, 1997; Krone and Snelting, 1994). Interestingly, we can observe an incipient interest on both "Annotation" and "Clone" approaches since 2012 with a share of 4% and 7%, respectively. Although they have been criticized due to its lack of modularity, these approaches have being the subject of recent efforts to overcome this limitation.

5.3. RQ3: Which kind of SPL asset received more attention and how is this attention evolving?

From the accumulated results in Fig. 5.4,[13] we notice that both the variability model (30%), and the code assets (30%) are the artefacts most addressed. This stems from the way we classified studies. Although studies might deal with distinct SPL assets (e.g., feature-to-code mappings, test assets, etc.), here we are interested in the assets that first evolve ("the subject of evolution"), rather

[12] This includes studies on external forces (for "Identify change"), variability-model analyses, metrics and negotiation processes (for "Analysis and plan change"), and change synchronization outside code assets (for "Implement change") and inconsistency checking of variability models (for "Verify change").

[13] "NA" (9%) refer to studies that consider no asset (e.g., a requirement prioritization algorithm (Inoki et al., 2014), monitoring the SPL environment to identify new needs (Böckle, 2005), etc).

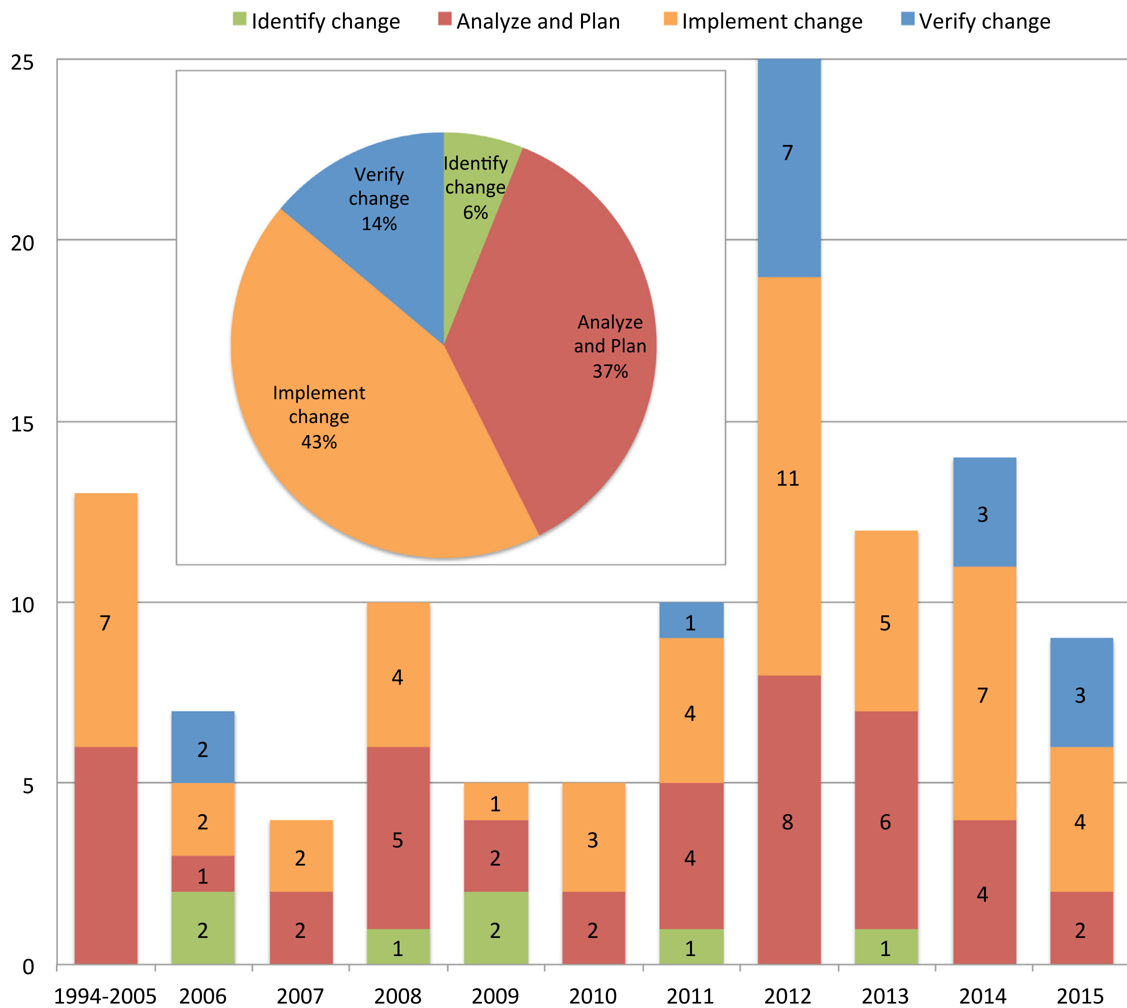

Fig. 5.5. "Evolution activity" over time.

than those assets that evolve as a result of the evolution of other assets. The latter assets are not computed into this facet.

"Code assets" account for 30%. Note that this category also includes *models* as the code counterpart in model-driven SPLs. Regarding the evolution over time, "SPL architecture" received more attention during the first years. This aligns with the findings of Heradio et al. (2016). On the other hand, products lag behind other assets as for attention received (13%). Though some proponents regard products to be derived on the fly from core assets, the current state of affairs is that products are still in need of being customized, and hence, having a detached life-cycle from the SPL.

5.4. RQ4: Which activities of the evolution life-cycle received most coverage and how is this coverage evolving?

Fig. 5.5 depicts the rate for each evolution activity. Note that it is possible for a paper to be categorized into more than one activity. This happens in eight cases, which explains why the total amounts goes up to 115. From the accumulated results, we observe that "Implement change" (43%) and "Analyze and plan change" (37%) account for more than half of the studies. Conversely, "Identify change" and "Verify change" lag behind with a rate of 6% and 14%, respectively. These differences might be partially explained by SPL challenges being more related to analysis and implementation, while change identification in SPLs exhibits some resemblance with single-product engineering. The stacked bar chart shows a sustained interest in "Implement change" and "Analyze and Plan change" over the years, while "Verify change" has recently received more attention.

So far, activities are those of Yau's change mini-cycle (Yau et al., 1978). This mini-cycle applies to any software artefact. However, we wanted to zoom into the specific sub-activities SPL practitioners cared about. Based on the mapping of primary studies conducted in Section 4, we refined Yau's model along nine sub-activities (see Fig. 4.1). Next subsections provide a finer-grained analysis of those sub-activities.

5.4.1. Zooming into identify change

Fig. 5.6 highlights this activity as being the less addressed: seven studies. Among the different forces of change, product engineering is the force more broadly addressed (Carbon et al., 2008; Mende et al., 2008; Creff et al., 2012), including customers' changing needs (Savolainen and Kuusela, 2001; Villela et al., 2010). This might be so, due to the fact of SPL products being amenable to be promoted as core assets, a distinctive aspect not applicable to single systems. On the other hand, the forces of change exerted by domain engineers are not so different from those found in single systems, hence, introducing less novelty. This likeness might also explain the sole existence of a study looking into "the SPL environment" (i.e., competitors, market research and technology forecasts) as a driver for SPL evolution: (Böckle, 2005).

Fig. 5.6. A finer-grained classification for SPL "Evolution activity".

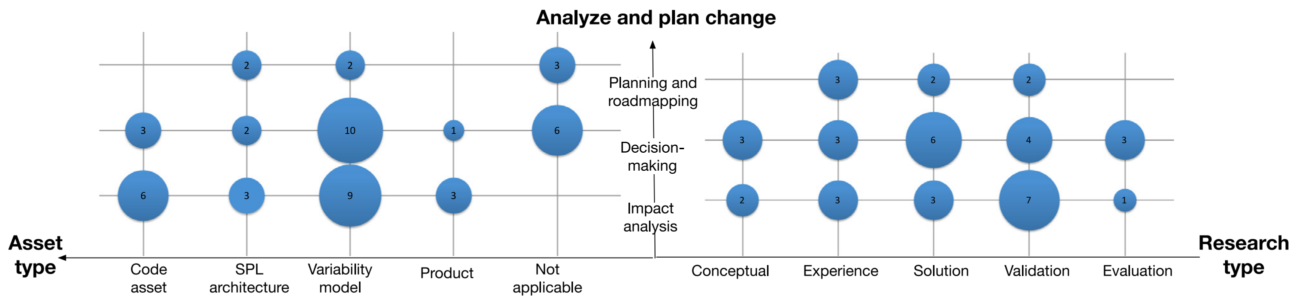

Fig. 5.7. Mapping "Analyze and plan change" across facets "Asset type" and "Research type".

5.4.2. Zooming into analyze and plan change

Fig. 5.6 depicts how "Decision-making" (17%) has received more attention that its siblings "Change impact" (14%) and "Planning" (6%). This might stem from SPLs bringing a new range of decisions concerning how assets evolve along the re-use spectrum. For these sub-activities, we are interested in finding what is the focus (i.e., facet "Asset type") and maturity (i.e., facet "Research type"). To this end, we crossed the activity dimension with these two facets. Fig. 5.7 depicts the outcome.

Impact analysis. Maturity level of CIA reveals that proposed techniques are mostly validated within academic case studies or experiments conducted in labs. These studies have mainly considered "Code Assets" and "Variability models" as the evolving assets. Products lag behind. This might evidence that academia barely considers product-specific changes which is at odds with common practice in industry (Rabiser et al., 2007).

Decision-making. At first glance, figures suggest this to be a rather mature area with three studies reaching the evaluation stage. However, this first impression should be contrasted against the kind of artefact being addressed. "Variability model" is the most tackled asset with nine studies. This might well stem from the formality brought by variability models that facilitates formal reasoning. However, other assets are largely overlooked. Specifically, the decision about product specifics being promoted to SPL

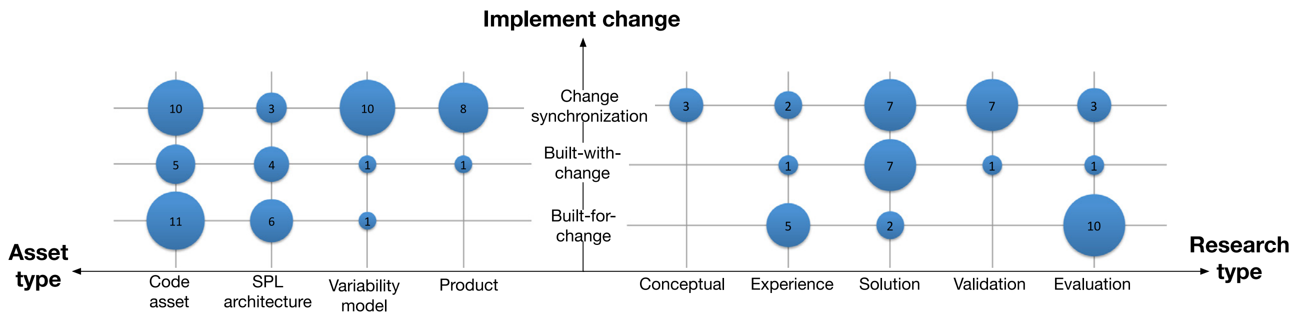

Fig. 5.8. Mapping "Implement change" across facets "Asset type" and "Research type".

core assets, has not received so much coverage despite being common in industry (Rabiser et al., 2007). This is an area that presumably will receive more attention in the future, especially if clone-based SPLs take off.

Planning and road-mapping. Studies seem to rely on industrial experiences to find evidences about how companies schedule and plan releases for SPLs. Variability models and the SPL architecture are the chosen artefacts for this kind of studies.

5.4.3. Zooming into implement change

Fig. 5.6 places "Change synchronization" (19%) as the most covered activity, ahead of "Built-for-change" (14%) and "Built-with-change" (9%). Next, we analyze each sub-activity w.r.t. asset focus and maturity (see Fig. 5.8).

Built-for-change. It comes as no surprise that code-artefact realization is by far the largest studied asset. It also stands out the comprehensive extent at which these studies have been conducted with nine studies reaching the evaluation stage.

Built-with-change. This sub-activity seems to mainly rely on "Solution proposals", and lacks empirical evaluation. Additionally, proposed approaches mostly aid engineers on performing changes at architecture and code asset level. Research on this field seems to underestimate product engineers when conducting product-specific changes (one study).

Change synchronization. This topic is receiving a steady interest in the last years. Special attention is devoted to keeping the SPL assets in sync along all abstraction levels, as well as, to keep synchronized SPL core assets and product assets. Specifically, "Evaluation research" has focused on keeping the variability model consistent with (smaller parts of) itself (Guo et al., 2012), and keeping in sync core assets and products (Dhungana et al., 2010; Heider et al., 2012a). The latter calls for effective configuration management approaches. We found several evidences at technical level, i.e., VCSs. For code assets, the trend seems to be to adapt new generation VCSs (e.g., BitKeeper, Git) to SPL's. However, we found neither experiences nor practices regarding how configuration management is achieved in industry.

5.4.4. Zooming into verify change

Fig. 5.6 gives a rough total for the sub-activities "Inconsistency detection" and "Scalable verification" of 9% and 5%, respectively. Mapping with the other dimensions indicates an evenly distribution of the studies w.r.t. both asset type and research type (see Fig. 5.9).

Inconsistency detection. Regarding the asset type, the variability model is the most addressed, presumably due to its readiness to formal reasoning. Specifically, Feature models are the favorite notation as opposed to Orthogonal Variability models, Decision-Oriented Variability models, or Cardinality-based models. Moreover, more than half of the studies include either validation or evaluation.

Scalable verification. Model checking is by far the most reported approach, and approaches to reduce re-verification effort upon changes, specially, on variability models and SPL architectures. Research in this field looks to be less mature compared to its sibling "inconsistency detection". This might be due to the difficulties in finding industrial cases where to test out the approaches.

6. Conclusions

This paper presented the results of a mapping study on SPL evolution. In total, 107 articles were included in this mapping study from 1994 to mid 2015. The aims were (1) to provide a consolidated overview on "SPL evolution", and (2), to identify well-established topics, trends and open research issues. As for the first goal, we described the SPL specifics and their impact on the traditional software change mini-cycle proposed by Yau et al. (1978). On these grounds, we further elaborated on this mini-cycle, and classified the literature accordingly. This permitted a finer grained classification of studies. The answers to the research question of our mapping study are presented below.

RQ1, Research type. Solution papers are the most common type of contribution (31%), followed by "Validation research" (24%). Nevertheless, a tendency can be observed towards more evaluation and validation papers. This admits a twofold interpretation: practitioners taking a more active role, or the research community becoming more mature. The area reaches a peak in 2012 with 25 papers, and it maintains a steady contribution of around 10 papers a year. Finally, four main conferences stand out as the main venues though SPLC takes the lion's share with a 28%.

RQ2, Product derivation approach. Efforts go as follows: "Annotation" (4%), "Clone" (7%), "Hybrid" (8%), "Model-driven" (15%), and "Composition-based" (29%), the later specially for component-based SPLs. Studies on FOP, AOP or DOP took the form of academic evaluations aiming at proving their resiliency upon SPL evolution. No evidences were found on the applicability of these approaches in an industrial setting. Interestingly, we observed a recent interest in both "Annotation" and "Clone" approaches since 2012 on.

RQ3, Asset type. Basically, all assets received coverage: variability model (30%), SPL architecture (18%), code assets (30%) and SPL products (13%). Products lag behind other assets as for attention received. Though some proponents regard products to be derived on the fly from core assets, the current state of affairs is that products are still in need of being customized, and hence, having their detached life-cycle from the SPL. This advices for products to be kept in the radar of SPL evolution.

RQ4, Evolution activity. "Identify change" and "Verify change" have received less attention (6% and 14%, respectively) compared to "Analyze and plan change" and "Implement change" (37% and 43%, respectively). A finer-grained analysis uncovered some tasks as being underexposed, namely, (1) decision-making on whether product specifics should be promoted to SPL core assets; (2) change impact analysis upon architectural changes; (3) inconsistency detec-

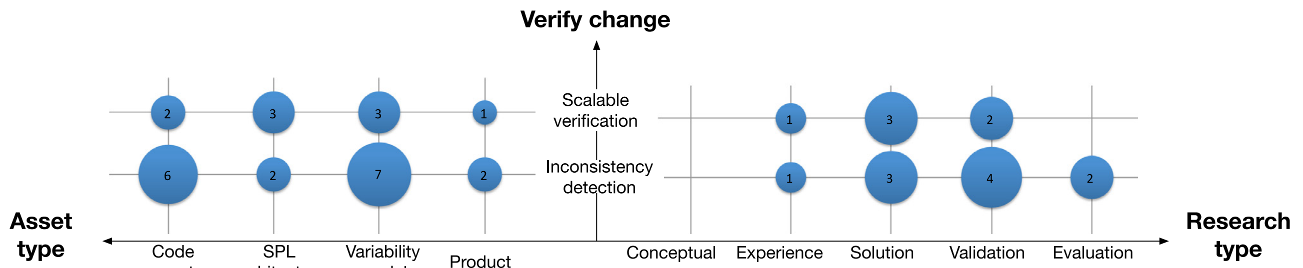

Fig. 5.9. Mapping "Verify change" across facets "Asset type" and "Research type".

tion for assets other than variability models. "Document change" was left out since no study was found on this activity.

From the results of this systematic mapping, we observe that SPLs are receiving considerably attention by the Software Engineering community with conferences fully dedicated to this topic. The increasing focus on evolution might be a symptom of maturity where SPL solutions start being tested out. Surprisingly, the number of "Experience papers" is rather limited (17%) which contrasts with the increasing use of SPLs in industry (Savolainen, 2013). A plea is then for practitioners to report their SPL evolution efforts. This would certainly be a spur for the whole field.

Acknowledgment

Thanks are due to authors of papers (Heider et al., 2012b; Sabouri and Khosravi, 2011; Michalik et al., 2011; Liu et al., 2007; Borba et al., 2012; Leopoldo Teixeira and Gheyi, 2015; Ribeiro et al., 2014; Mende et al., 2008; Loughran and Rashid, 2004; Rumpe et al., 2015; Lity et al., 2012; Vierhauser et al., 2012; Demuth et al., 2014; Tizzei et al., 2011; Peng et al., 2011; Michalik and Weyns, 2011; Sabouri and Khosravi, 2014; Cordy et al., 2012; da Mota Silveira Neto et al., 2012; Garg et al., 2003; Käßmeyer et al., 2015; ter Beek et al., 2012; Júnior and Coelho, 2011; Inoki et al., 2014; Pleuss et al., 2012; Rubin et al., 2012; Teixeira et al., 2015; Díaz et al., 2014) who promptly helped us to better classify their work. This work is co-supported by the Spanish Ministry of Education, and the European Social Fund under contract TIN2014-58131-R. Montalvillo enjoys a doctoral grant from the University of the Basque Country.

Appendix A. Included papers classified on facets

Table A.1
Primary study facet classification.

Ref.	Title	Year	Evolution activity	Evolution sub-activity	Asset type	Product-derivation appr.	Research type
Käßmeyer et al. (2015)	A process to support a systematic change impact analysis of variability and safety in automotive functions	2015	Analyze and plan	Change impact	Products	Model-driven	Solution
Heider et al. (2012c)	A case study on the evolution of a component-based product line	2012	Analyze and plan	Change impact	Variability model, SPL architecture, Code assets	Composition	Experience
Schackmann and Lichter (2006)	A cost-based approach to software product line management	2006	Analyze and plan	Decision-making	Variability model	NA	Conceptual
Barreiros and Moreira (2014)	A cover-based approach for configuration repair	2014	Implement	Change synchro-nization	Variability model	NA	Validation
Murthy et al. (1994)	A holistic approach to product marketability measurements-the PMM approach	1994	Analyze and plan	Decision-making	Products	NA	Solution
Thurimella and Brügge (2013)	A mixed-method approach for the empirical evaluation of the issue-based variability modeling	2013	Analyze and plan	Decision-making	Variability model	NA	Evaluation
Leopoldo Teixeira and Gheyi (2015)	A product line of theories for reasoning about safe evolution of product lines	2015	Implement Verify	Built-with-change Inconsistency checking	Variability model, SPL architecture, Code assets	Hybrid	Solution
Gaia et al. (2014)	A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines	2014	Implement	Built-for-change	Code assets	Composition	Evaluation
Schmid and Eichelberger (2007)	A requirements-based taxonomy of software product line evolution	2007	Identify	Monitoring the environment	NA	NA	Conceptual
Borba et al. (2012)	A theory of software product line refinement	2012	Implement Verify	Built-with-change Inconsistency checking	Variability model, Code assets	Hybrid	Solution
Deng et al. (2005)	Addressing domain evolution challenges in software product lines	2006	Implement	Built-for-change	SPL architecture	Model-driven	Solution

(continued on next page)

Table A.1 (*continued*)

Ref.	Title	Year	Evolution activity	Evolution sub-activity	Asset type	Product-derivation appr.	Research type
Díaz et al. (2014)	Agile product-line architecting in practice: A case study in smart grids	2014	Analyze and plan Implement	Change impact Built-for-change	SPL architecture	Composition	Evaluation
Noor et al. (2008)	Agile product line planning: A collaborative approach and a case study	2008	Analyze and plan	Planning	NA	NA	Validation
Corrêa et al. (2011)	An analysis of change operations to achieve consistency in model-driven software product lines	2011	Implement	Change synchronization	Code assets	Model-driven	Conceptual
Sarang and Sanglikar (2007)	An analysis of effort variance in software maintenance projects	2008	Analyze and plan	Decision-making	Code assets, SPL architecture, Code assets	NA	Solution
Tran and Massacci (2014)	An approach for decision support on the uncertainty in feature model evolution	2014	Analyze and plan	Decision-making	Variability model	NA	Solution
Garg et al. (2003)	An environment for managing evolving product line architectures	2003	Implement	Built-with-change	SPL architecture	Composition	Solution
da Mota Silveira Neto et al. (2012)	An experimental study to evaluate a SPL architecture regression testing approach	2012	Verify	Inconsistency checking	SPL architecture, Code assets	Composition	Evaluation
Cafeo et al. (2012)	Analysing the impact of feature dependency implementation on product line stability: An exploratory study	2012	Implement	Built-for-change	Variability model, Code assets	NA	Evaluation
Peng et al. (2011)	Analyzing evolution of variability in a software product line: From contexts and requirements to features	2011	Analyze and plan	Decision-making	Variability model	NA	Conceptual
Inoki et al. (2014)	Application of requirements prioritization decision rules in software product line evolution	2014	Analyze and plan	Planning	NA	Composition	Experience
Vierhauser et al. (2012)	Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines	2012	Verify	Inconsistency checking	Variability model, Code assets	Model-driven	Evaluation
Dikel et al. (1997)	Applying software product-line architecture	1997	Implement	Built-for-change	SPL architecture	NA	Experience
Gámez and Fuentes (2013)	Architectural evolution of FamiWare using cardinality-based feature models	2013	Analyze and plan	Decision-making	Variability model	NA	Validation
			Implement	Change synchronization			
Menkyna and Vranic (2009)	Aspect-oriented change realization based on multi-paradigm design with feature modeling	2012	Implement	Built-with-change	Code assets	Composition	Solution
ter Beek et al. (2012)	Assume-guarantee testing of evolving software product line architectures	2012	Verify	Scalable verification	SPL architecture	Composition	Solution
Demuth et al. (2014)	Automatic and incremental product optimization for software product lines	2014	Verify	Inconsistency checking	Variability model, Code assets	Composition	Validation
Rumpe et al. (2015)	Behavioral compatibility of simulink models for product line maintenance and evolution	2015	Verify	Scalable verification	SPL architecture	Composition	Validation
Karimpour and Ruhe (2013)	Bi-criteria genetic search for adding new features into an existing product line	2013	Analyze and plan	Decision-making	Variability model, Code assets	NA	Solution
Holdschick (2012)	Challenges in the evolution of model-based software product lines in the automotive domain	2012	Implement	Change synchronization	SPL architecture	Composition	Experience
Paskevicius et al. (2012)	Change impact analysis of feature models	2012	Analyze and plan	Change impact	Variability model	NA	Solution
Seidl et al. (2012)	Co-evolution of models and feature mapping in software product lines	2012	Implement	Change synchronization	Variability model, Code assets	Model-driven	Solution
Passos et al. (2013)	Coevolution of variability models and related artifacts: A case study from the Linux kernel	2013	Implement	Change synchronization	Variability model	Annotation	Validation
Cardone and Lin (2001)	Comparing frameworks and layered refinement	2001	Implement	Built-for-change	Code assets	Hybrid	Evaluation
Abdelmoez et al. (2012)	Comparing maintainability evolution of object-oriented and aspect-oriented software product lines	2012	Implement	Built-for-change	Code assets	Composition	Evaluation
Tizzei et al. (2011)	Components meet aspects: Assessing design stability of a software product line	2011	Implement	Built-for-change	SPL architecture	Composition	Evaluation

(*continued on next page*)

Table A.1 (*continued*)

Ref.	Title	Year	Evolution activity	Evolution sub-activity	Asset type	Product-derivation appr.	Research type
Quinton et al. (2014)	Consistency checking for the evolution of cardinality-based feature models	2014	Verify	Inconsistency checking	Variability model	NA	Validation
Guo et al. (2012)	Consistency maintenance for evolving feature models	2012	Implement	Change synchro-nization	Variability model	NA	Evaluation
Lity et al. (2012)	Delta-oriented model-based SPL regression testing	2012	Verify	Scalable verification	Products	Compositon	Solution
Schaefer et al. (2010)	Delta-oriented programming of software product lines	2010	Implement	Built-for-change	Code assets	Composition	Evaluation
Tischer et al. (2012)	Developing long-term stable product line architectures	2012	Implement	Built-for-change	SPL architecture	NA	Experience
Chen et al. (2003)	Differencing and merging within an evolving product line architecture	2004	Implement	Change synchro-nization	Products	Composition	Solution
Dintzner et al. (2015)	Evaluating feature change impact on multi-product line configurations using partial information	2015	Analyze and plan	Change impact	Variability model	NA	Validation
Villela et al. (2010)	Evaluation of a method for proactively managing the evolving scope of a software product line	2010	Identify	Monitoring customer	NA	NA	Evaluation
Thurimella and Bruegge (2007)	Evolution in product line requirements engineering: A rationale management approach	2007	Analyze and plan	Decision-making	NA	NA	Validation
Pichler et al. (2011)	Evolution patterns for business document models	2011	Analyze and plan	Change impact	Variability model	Model-driven	Solution
Tesanovic (2007)	Evolving embedded product lines: Opportunities for aspects	2007	Implement	Built-for-change	Code assets	Composition	Experience
Figueiredo et al. (2008)	Evolving software product lines with aspects	2008	Implement	Built-for-change	Code assets	Composition	Evaluation
Riva and Rosso (2003)	Experiences with software product family evolution	2003	Analyze and plan	Decision-making	SPL architecture	NA	Experience
Sharp (1999)	Exploiting object technology to support product variability	1999	Implement	Built-for-change	Code assets	Hybrid	Experience
Heider et al. (2012a)	Facilitating the evolution of products in product line engineering by capturing and replaying configuration decisions	2012	Implement	Change synchro-nization	Variability model, Code assets	Model-driven	Evaluation
Ribeiro et al. (2014)	Feature maintenance with emergent interfaces	2014	Analyze and plan Implement	Change impact built-with-change	Code assets	Annotation	Validation
Schröter et al. (2014)	Feature-context interfaces: Tailored programming interfaces for software product lines	2014	Implement	Built-with-change	Code assets	Composition	Evaluation
Yazdanshenas and Moonen (2012)	Fine-grained change impact analysis for component-based product families	2012	Analyze and plan	Change impact	Code assets	Composition	Validation
Jarzabek and Trung (2011)	Flexible generators for software reuse and evolution	2011	Implement	Built-with-change	Products	Model-driven	Solution
Antkiewicz et al. (2014)	Flexible product line engineering with virtual platform	2014	Implement	Change synchro-nization	Products	Clone	Conceptual
Loughran and Rashid (2004)	Framed Aspects: supporting variability and configurability for AOP	2009	Implement	Built-for-change	Code assets	Hybrid	Solution
Taborda (2004)	Generalized release planning for product line architectures	2004	Analyze and plan	Planning	SPL architecture	NA	Experience
Thurimella et al. (2008)	Identifying and exploiting the similarities between rationale management and variability management	2008	Analyze and plan	Decision-making	Variability model	NA	Evaluation
Anastasopoulos (2009)	Increasing efficiency and effectiveness of software product line evolution: An infrastructure on top of configuration management	2009	Implement	Change synchro-nization	Variability model, Code assets, Products	Composition	Validation
Böckle (2005)	Innovation management for product line engineering organizations	2005	Identify	Monitoring the environment	NA	NA	Conceptual
Thurimella and Bruegge (2012)	Issue-based variability management	2012	Analyze and plan	Decision-making	Variability model	NA	Evaluation
Svahnberg and Bosch (2000)	Issues concerning variability in software product lines	2000	Implement	Built-for-change	SPL architecture	Hybrid	Experience
Livengood (2011)	Issues in software product line evolution: complex changes in variability models	2011	Analyze and plan	Change impact	Variability model	Annotation	Experience
Dyer et al. (2013)	Language features for software evolution and aspect-oriented interfaces: An exploratory study	2013	Implement	Built-for-change	Code assets	Composition	Evaluation
Ji et al. (2015)	Maintaining feature traceability with embedded annotations	2015	Implement	Change synchro-nization	Variability model	Clone	Validation

(*continued on next page*)

Table A.1 (*continued*)

Ref.	Title	Year	Evolution activity	Evolution sub-activity	Asset type	Product-derivation appr.	Research type
Annosi et al. (2012)	Managing and assessing the risk of component upgrades	2012	Analyze and plan	Decision-making	Code assets	Composition	Experience
Jiang et al. (2008)	Maintaining software product lines: an industrial practice	2008	Analyze and plan	Change impact	Code assets	Composition	Experience
Rubin et al. (2013)	Managing cloned variants : A Framework and experience	2013	Analyze and plan Implement	Change impact Change synchro-nization	Products	Clone	Validation
Cordy et al. (2012)	Managing evolution in software product lines: A model-checking perspective	2012	Verify	Scalable verification	Variability model, Code assets	Model-driven	Solution
Rubin et al. (2012)	Managing forked product variants	2012	Analyze and plan Implement	Change impact Change synchro-nization	Products	Clone	Conceptual
Creff et al. (2012)	Model-based product line evolution: An incremental growing by extension	2012	Identify	Monitoring products	Products	Model-driven	Solution
Schubanz et al. (2013)	Model-driven planning and monitoring of long-term software product line evolution	2013	Analyze and plan	Planning	Variability model	Model-driven	Solution
Pleuss et al. (2012)	Model-driven support for product line evolution on feature level	2012	Analyze and plan	Planning	Variability model	Model-driven	Validation
Hendrickson and van der Hoek (2007)	Modeling product line architectures through change sets and relationships	2007	Implement	Built-with-change	SPL architecture	Composition	Solution
Heider et al. (2010b)	Negotiation constellations in reactive product line evolution	2010	Analyze and plan	Decision-making	NA	NA	Conceptual
Michalik et al. (2011)	On the problems with evolving egemin's software product line	2011	Implement	Change synchro-nization	SPL architecture	Composition	Experience
Ferreira et al. (2014)	On the use of feature-oriented programming for evolving software product lines - A comparative study	2014	Implement	Built-for-change	Code assets	Hybrid	Evaluation
Scheidemann (2006)	Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems	2006	Verify	Scalable verification	Variability model, SPL architecture	NA	Experience
McVoy (2015)	Preliminary product line support in BitKeeper	2015	Implement	Change synchro-nization	Code assets	Clone	Solution
Júnior and Coelho (2011)	Preserving the exception handling design rules in software product line context: A practical approach	2011	Verify	Inconsistency checking	Code assets	Composition	Validation
Kakarontzas et al. (2008)	Product line variability with elastic components and test-driven development	2008	Implement	Built-with-change	Products	Composition	Solution
Carbon et al. (2008)	Providing feedback from application to family engineering: The product line planning game at the Testo AG	2008	Identify	Monitoring products	Products	NA	Experience
Thüm et al. (2009)	Reasoning about edits to feature models	2009	Analyze and plan	Change impact	Variability model	NA	Validation
Dhaliwal et al. (2012)	Recovering commit dependencies for selective code integration in software product line	2012	Implement	Change synchro-nization	Code assets	Composition	Validation
Sabouri and Khosravi (2014)	Reducing the verification cost of evolving product families using static analysis techniques	2014	Verify	Scalable verification	Variability model, Code assets	Annotation	Validation
van Ommering (2001)	Roadmapping a product population architecture	2002	Analyze and plan	Planning	SPL architecture	Composition	Solution
Teixeira et al. (2015)	Safe evolution of product populations and multi product lines	2015	Verify	Inconsistency checking	Variability model, Code assets	NA	Solution
Savolainen and Kuusela (2008)	Scheduling product line features for effective roadmapping	2008	Analyze and plan	Planning	NA	NA	Experience
Heider et al. (2010a)	Simulating evolution in model-based product line engineering	2010	Analyze and plan	Decision-making	NA	NA	Validation
Thao et al. (2008)	Software Configuration Management for Product Derivation in Software Product Families	2008	Implement	Change synchro-nization	Variability model, Code assets, Products	Composition	Solution
Romero et al. (2013)	SPLEMMA: A generic framework for controlled-evolution of software product lines	2013	Implement	Built-with-change	Variability model	NA	Solution
Liu et al. (2007)	State-based modeling to support the evolution and maintenance of safety-critical software product lines	2007	Analyze and plan	Decision-making	NA	NA	Solution

(*continued on next page*)

Table A.1 (*continued*)

Ref.	Title	Year	Evolution activity	Evolution sub-activity	Asset type	Product-derivation appr.	Research type
Knodel et al. (2006)	Static evaluation of software architectures	2006	Implement	Built-with-change	SPL architecture	NA	Experience
			Verify	Inconsistency checking			
Dhungana et al. (2010)	Structuring the modeling space and supporting evolution in software product line engineering	2010	Implement	Change synchronization	Variability model, Code assets	Model-driven	Evaluation
Jahn et al. (2012)	Supporting model maintenance in component-based product lines	2012	Verify	Inconsistency checking	Variability model, Code assets	Model-driven	Validation
Mende et al. (2008)	Supporting the grow-and-prune model in software product lines evolution using clone detection	2008	Identify	Monitoring products	Products	Clone	Evaluation
Shen et al. (2010)	Synchronized architecture evolution in software product line using bidirectional transformation	2010	Implement	Change synchronization	SPL architecture, Products	Model-driven	Solution
Kim and Czarnecki (2005)	Synchronizing cardinality-based feature models and their specializations	2005	Implement	Change synchronization	Variability model	NA	Solution
Schmid and Verlage (2002)	The economic impact of product line adoption and evolution	2002	Analyze and plan	Decision-making	NA	NA	Experience
Michalik and Weyns (2011)	Towards a solution for change impact analysis of software product line products	2011	Analyze and plan	Change impact	Variability model, SPL architecture, Code assets	Model-driven	Conceptual
Montalvillo and Díaz (2015)	Tunning Github for SPL development: branching models and operations for product engineers	2015	Implement	Change synchronization	Code assets, Products	Composition	Solution
Heider et al. (2012b)	Using regression testing to analyze the impact of changes to variability models on products	2012	Analyze and plan	Change impact	Variability model	Model-driven	Validation
Chen et al. (2004)	Using simulation to facilitate the study of software product line evolution	2004	Analyze and plan	Decision-making	NA	NA	Solution
Deelstra et al. (2009)	Variability assessment in software product families	2009	Analyze and plan	Decision-making	Variability model	NA	Validation
Savolainen and Kuusela (2001)	Violatility analysis framework for product lines	2001	Identify	Monitoring customer	NA	NA	Solution
Murashkin et al. (2013)	Visualization and exploration of optimal variants in product line engineering	2013	Analyze and plan	Change impact	Variability model	NA	Validation

References

Abdelmoez, W., Khater, H., El-shoafy, N., 2012. Comparing maintainability evolution of object-oriented and aspect-oriented software product lines. In: 18th International Conference on informatics and Systems (INFOS 2012), pp. 53–60.

Abdelmoez, W., Nassar, D.E.M., Shereshevsky, M., Gradetsky, N., Gunnalan, R., Ammar, H.H., Yu, B., Mili, A., 2004. Error propagation in software architectures. In: 10th IEEE International Software Metrics Symposium (METRICS), 11–17 September 2004, Chicago, IL, USA, pp. 384–393. doi:10.1109/METRIC.2004.1357923.

Ahn, S., Chong, K., 2007. Requirements change management on feature-oriented requirements tracing. In: Computational Science and Its Applications - ICCSA 2007, International Conference, Kuala Lumpur, Malaysia, August 26–29, 2007. Proceedings, Part II, pp. 296–307. doi:10.1007/978-3-540-74477-1_29.

Ajila, S., Dumitrescu, R.T., 2007. Experimental use of code delta, code churn, and rate of change to understand software product line evolution. J. Syst. Softw. 74–91. doi:10.1016/j.jss.2006.05.034.

Ajila, S.A., Kaba, A.B., 2008. Evolution support mechanisms for software product line process. J. Syst. Softw. 81 (10), 1784–1801. doi:10.1016/j.jss.2007.12.797.

Alves, V., Calheiros, F., Nepomuceno, V., Menezes, A., Soares, S., Borba, P., 2008. Flip: Managing software product line extraction and reaction with aspects. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8–12, 2008, Proceedings, p. 354. doi:10.1109/SPLC.2008.51.

Alves, V., Gheyi, R., Massoni, T., Kulesza, U., Borba, P., de Lucena, C.J.P., 2006. Refactoring product lines. In: Generative Programming and Component Engineering, 5th International Conference, GPCE 2006, Portland, Oregon, USA, October 22–26, 2006, Proceedings, pp. 201–210. doi:10.1145/1173706.1173737.

Anastasopoulos, M., 2009. Increasing efficiency and effectiveness of software product line evolution: an infrastructure on top of configuration management. In: Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops, Amsterdam, Netherlands, August 24–28, 2009, pp. 47–56. doi:10.1145/1595808.1595819.

Annosi, M.C., Penta, M.D., Tortora, G., 2012. Managing and assessing the risk of component upgrades. In: Proceedings of the Third International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2012, Zurich, Switzerland, June 4, 2012, pp. 9–12. doi:10.1109/PLEASE.2012.6229776.

Anquetil, N., Kulesza, U., Mitschke, R., Moreira, A., Royer, J., Rummler, A., Sousa, A., 2010. A model-driven traceability framework for software product lines. Softw. Syst. Model. 9 (4), 427–451. doi:10.1007/s10270-009-0120-9.

Antkiewicz, M., Ji, W., Berger, T., Czarnecki, K., Schmorleiz, T., Lämmel, R., Stanciulescu, S., Wasowski, A., Schaefer, I., 2014. Flexible product line engineering with a virtual platform. In: 36th International Conference on Software Engineering, ICSE '14, Companion Proceedings, Hyderabad, India, May 31, - June 07, 2014, pp. 532–535. doi:10.1145/2591062.2591126.

Apel, S., Batory, D.S., Kästner, C., Saake, G., 2013. Feature-Oriented Software Product Lines - Concepts and Implementation. Springer doi:10.1007/978-3-642-37521-7.

Barney, S., Petersen, K., Svahnberg, M., Aurum, A., Barney, H.T., 2012. Software quality trade-offs: A systematic map. Inf. Softw. Technol. 54 (7), 651–662. doi:10.1016/j.infsof.2012.01.008.

Barreiros, J., Moreira, A., 2014. A cover-based approach for configuration repair. In: 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15–19, 2014, pp. 157–166. doi:10.1145/2648511.2648528.

Batory, D.S., Sarvela, J.N., Rauschmayer, A., 2004. Scaling step-wise refinement. IEEE Trans. Softw. Eng. 30 (6), 355–371. doi:10.1109/TSE.2004.23.

Bennett, K.H., Rajlich, V., 2000. Software maintenance and evolution: a roadmap. In: 22nd International Conference on Software Engineering, Future of Software Engineering Track, ICSE 2000, Limerick Ireland, June 4–11, 2000., pp. 73–87. doi:10.1145/336512.336534.

Berezin, S., Campos, S.V.A., Clarke, E.M., 1997. Compositional reasoning in model checking. In: Compositionality: The Significant Difference, International Symposium, COMPOS'97, Bad Malente, Germany, September 8–12, 1997. Revised Lectures, pp. 81–102. doi:10.1007/3-540-49213-5_4.

Bertran, I.M., Garcia, A., von Staa, A., 2010. Defining and applying detection strategies for aspect-oriented code smells. In: 24th Brazilian Symposium on Software Engineering, SBES 2010, Salvador, Bahia, Brazil, September 27, - October 1, 2010, pp. 60–69. doi:10.1109/SBES.2010.14.

Beuche, D., Papajewski, H., Schröder-Preikschat, W., 2004. Variability management with feature models. Sci. Comput. Program. 53 (3), 333–352. doi:10.1016/j.scico.2003.04.005.

Böckle, G., 2005. Innovation management for product line engineering organizations. In: Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26–29, 2005, Proceedings, pp. 124–134. doi:10.1007/11554844_14.

Boehm, B.W., Bose, P.K., Horowitz, E., Lee, M.J., 1994. Software requirements as negotiated win conditions. In: Proceedings of the First IEEE International Conference on Requirements Engineering, ICRE '94, Colorado Springs, Colorado, USA, April 18–21, 1994, pp. 74–83. doi:10.1109/ICRE.1994.292400.

Bohner, S.A., 1996. Impact analysis in the software change process: a year 2000 perspective. In: 1996 International Conference on Software Maintenance (ICSM '96), 4–8 November 1996, Monterey, CA, USA, Proceedings, pp. 42–51. doi:10.1109/ICSM.1996.564987.

Borba, P., Teixeira, L., Gheyi, R., 2012. A theory of software product line refinement. Theor. Comput. Sci. 455, 2–30. doi:10.1016/j.tcs.2012.01.031.

Bosch, J., 2002. Maturity and evolution in software product lines: Approaches, artefacts and organization. In: Software Product Lines, Second International Conference, SPLC 2, San Diego, CA, USA, August 19–22, 2002, Proceedings, pp. 257–271. doi:10.1007/3-540-45652-X_16.

Botterweck, G., Pleuss, A., 2014. Evolution of software product lines. In: Evolving Software Systems, pp. 265–295. doi:10.1007/978-3-642-45398-4_9.

Botterweck, G., Pleuss, A., Polzer, A., Kowalewski, S., 2009. Towards feature-driven planning of product-line evolution. In: Proceedings of the First International Workshop on Feature-Oriented Software Development, FOSD 2009, Denver, Colorado, USA, October 6, 2009, pp. 109–116. doi:10.1145/1629716.1629737.

Breivold, H.P., Larsson, S., Land, R., 2008. Migrating industrial systems towards software product lines: Experiences and observations through case studies. In: 34th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2008, September 3–5, 2008, Parma, Italy, pp. 232–239. doi:10.1109/SEAA.2008.13.

Budgen, D., Turner, M., Brereton, P., Kitchenham, B.A., 2008. Using mapping studies in software engineering. In: Proceedings of Psychology of Programming Interest Group (PPIG), 8, pp. 195–204.

Cafeo, B.B.P., Dantas, F., Gurgel, A.C., Guimarães, E.T., Cirilo, E., Garcia, A.F., de Lucena, C.J.P., 2012. Analysing the impact of feature dependency implementation on product line stability: an exploratory study. In: 26th Brazilian Symposium on Software Engineering, SBES 2012, Natal, Brazil, September 23–28, 2012, pp. 141–150. doi:10.1109/SBES.2012.23.

Capilla, R., Bosch, J., Trinidad, P., Cortés, A.R., Hinchey, M., 2014. An overview of dynamic software product line architectures and techniques: observations from research and industry. J. Syst. Softw. 91, 3–23. doi:10.1016/j.jss.2013.12.038.

Carbon, R., Knodel, J., Muthig, D., Meier, G., 2008. Providing feedback from application to family engineering - the product line planning game at the testo AG. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8–12, 2008, Proceedings, pp. 180–189. doi:10.1109/SPLC.2008.21.

Cardone, R., Lin, C., 2001. Comparing frameworks and layered refinement. In: Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12–19 May 2001, Toronto, Ontario, Canada, pp. 285–294. doi:10.1109/ICSE.2001.919102.

Casteleyn, S., Garrigós, I., Mazón, J., 2014. Ten years of rich internet applications: asystematic mapping study, and beyond. TWEB 8 (3), 18:1–18:46. doi:10.1145/2626369.

Chen, L., Babar, M.A., 2011. A systematic review of evaluation of variability management approaches in software product lines. Inf. Softw. Technol. 53 (4), 344–362. doi:10.1016/j.infsof.2010.12.006.

Chen, P., Critchlow, M., Garg, A., van der Westhuizen, C., van der Hoek, A., 2003. Differencing and merging within an evolving product line architecture. In: Software Product-Family Engineering, 5th International Workshop, PFE 2003, Siena, Italy, November 4–6, 2003, Revised Papers, pp. 269–281. doi:10.1007/978-3-540-24667-1_20.

Chen, Y., Gannod, G.C., Collofello, J.S., Sarjoughian, H.S., 2004. Using simulation to facilitate the study of software product line evolution. In: 7th International Workshop on Principles of Software Evolution (IWPSE 2004), 6–7 September 2004, Kyoto, Japan, pp. 103–112. doi:10.1109/IWPSE.2004.1334774.

Clements, P., Northrop, L., 2001. Software Product Lines: Practices and Patterns. Addison-Wesley Professional.

Cordy, M., Classen, A., Schobbens, P., Heymans, P., Legay, A., 2012. Managing evolution in software product lines: a model-checking perspective. In: Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25–27, 2012. Proceedings, pp. 183–191. doi:10.1145/2110147.2110168.

Corrêa, C.K.F., de Oliveira, T.C., Werner, C.M.L., 2011. An analysis of change operations to achieve consistency in model-driven software product lines. In: Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22–26, 2011. Workshop Proceedings (Volume 2), p. 24. doi:10.1145/2019136.2019163.

Creff, S., Champeau, J., Jézéquel, J., Monégier, A., 2012. Model-based product line evolution: an incremental growing by extension, 107–114.

Czarnecki, K., 2007. Software reuse and evolution with generative techniques. In: 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5–9, 2007, Atlanta, Georgia, USA, p. 575. doi:10.1145/1321631.1321750.

da Mota Silveira Neto, P. A., do Carmo Machado, I., Cavalcanti, Y. C., de Almeida, E. S., Garcia, V. C., de Lemos Meira, S. R., 2012. An experimental study to evaluate a SPL architecture regression testing approach, 608–615. 10.1109/IRI.2012.6303065

da Mota Silveira Neto, P.A., do Carmo Machado, I., McGregor, J.D., de Almeida, E.S., de Lemos Meira, S.R., 2011. A systematic mapping study of software product lines testing. Inf. Softw. Technol. 53 (5), 407–423. doi:10.1016/j.infsof.2010.12.003.

Deelstra, S., Sinnema, M., Bosch, J., 2005. Product derivation in software product families: a case study. J. Syst. Softw. 74 (2), 173–194. doi:10.1016/j.jss.2003.11.012.

Deelstra, S., Sinnema, M., Bosch, J., 2009. Variability assessment in software product families. Inf. Softw. Technol. 51 (1), 195–218. doi:10.1016/j.infsof.2008.04.002.

Delaware, B., Cook, W.R., Batory, D.S., 2009. Fitting the pieces together: a machine-checked model of safe composition. In: Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24–28, 2009, pp. 243–252. doi:10.1145/1595696.1595733.

Demuth, A., Lopez-Herrejon, R.E., Egyed, A., 2014. Automatic and incremental product optimization for software product lines. In: Seventh IEEE International Conference on Software Testing, Verification and Validation, ICST 2014, March 31 2014-April 4, 2014, Cleveland, Ohio, USA, pp. 31–40. doi:10.1109/ICST.2014.14.

Deng, G., Lenz, G., Schmidt, D.C., 2005. Addressing domain evolution challenges in software product lines. In: Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Doctoral Symposium, Educators Symposium, Montego Bay, Jamaica, October 2–7, 2005, Revised Selected Papers, pp. 247–261. doi:10.1007/11663430_26.

Devine, T.R., Goseva-Popstojanova, K., Krishnan, S., Lutz, R.R., 2014. Assessment and cross-product prediction of software product line quality: accounting for reuse across products, over multiple releases. Autom. Softw. Eng. 23 (2), 253–302. doi:10.1007/s10515-014-0160-4.

Dhaliwal, T., Khomh, F., Zou, Y., Hassan, A.E., 2012. Recovering commit dependencies for selective code integration in software product lines. In: 28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23–28, 2012, pp. 202–211. doi:10.1109/ICSM.2012.6405273.

Dhungana, D., Grünbacher, P., Rabiser, R., Neumayer, T., 2010. Structuring the modeling space and supporting evolution in software product line engineering. J. Syst. Softw. 83 (7), 1108–1122. doi:10.1016/j.jss.2010.02.018.

Dhungana, D., Neumayer, T., Grúnbacher, P., Rabiser, R., 2008. Supporting evolution in model-based product line engineering. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8–12, 2008, Proceedings, pp. 319–328. doi:10.1109/SPLC.2008.26.

Dhungana, D., Rabiser, R., Grünbacher, P., Neumayer, T., 2007. Integrated tool support for software product line engineering. In: 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5–9, 2007, Atlanta, Georgia, USA, pp. 533–534. doi:10.1145/1321631.1321730.

Díaz, J., Pérez, J., Garbajosa, J., 2014. Agile product-line architecting in practice: a case study in smart grids. Inf. Softw. Technol. 56 (7), 727–748. doi:10.1016/j.infsof.2014.01.014.

Dikel, D., Kane, D., Ornburn, S., Loftus, W., Wilson, J., 1997. Applying software product-line architecture. IEEE Comput. 30 (8), 49–55. doi:10.1109/2.607064.

Dintzner, N., Kulesza, U., van Deursen, A., Pinzger, M., 2015. Evaluating feature change impact on multi-product line configurations using partial information. In: Software Reuse for Dynamic Systems in the Cloud and Beyond - 14th International Conference on Software Reuse, ICSR 2015, Miami, FL, USA, January 4–6, 2015. Proceedings, pp. 1–16. doi:10.1007/978-3-319-14130-5_1.

do Carmo Machado, I., McGregor, J.D., Cavalcanti, Y.C., de Almeida, E.S., 2014. On strategies for testing software product lines: A systematic literature review. Inf. Softw. Technol. 56 (10), 1183–1199. doi:10.1016/j.infsof.2014.04.002.

Duszynski, S., Knodel, J., Lindvall, M., 2009. SAVE: software architecture visualization and evaluation. In: 13th European Conference on Software Maintenance and Reengineering, CSMR 2009, Architecture-Centric Maintenance of Large-SCale Software Systems, Kaiserslautern, Germany, 24–27 March 2009, pp. 323–324. doi:10.1109/CSMR.2009.52.

Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: asystematic review. Inf. Softw. Technol. 50 (9–10), 833–859. doi:10.1016/j.infsof.2008.01.006.

Dyer, R., Rajan, H., Cai, Y., 2013. Language features for software evolution and aspect-oriented interfaces: an exploratory study. T. Aspect-Oriented Softw. Dev. 10, 148–183. doi:10.1007/978-3-642-36964-3_5.

Engström, E., Runeson, P., 2010. A qualitative survey of regression testing practices. In: Product-Focused Software Process Improvement, 11th International Conference, PROFES 2010, Limerick, Ireland, June 21–23, 2010. Proceedings, pp. 3–16. doi:10.1007/978-3-642-13792-1_3.

Engström, E., Runeson, P., 2011. Software product line testing - A systematic mapping study. Inf. Softw. Technol. 53 (1), 2–13. doi:10.1016/j.infsof.2010.05.011.

Ernst, M.D., Badros, G.J., Notkin, D., 2002. An empirical analysis of C preprocessor use. IEEE Trans. Softw. Eng. 28 (12), 1146–1170. doi:10.1109/TSE.2002.1158288.

Favre, J., 1997. Understanding-in-the-large. In: 5th International Workshop on Program Comprehension (WPC '97), May 28–30, 1997 - Dearborn, MI, USA, pp. 29–38. doi:10.1109/WPC.1997.601260.

Ferreira, G.C.S., Gaia, F.N., Figueiredo, E., de Almeida Maia, M., 2014. On the use of feature-oriented programming for evolving software product lines - A comparative study. Sci. Comput. Program. 93, 65–85. doi:10.1016/j.scico.2013.10.010.

Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F.C., Khan, S.S., Filho, F.C., Dantas, F., 2008. Evolving software product lines with aspects: an empirical study on design stability. In: 30th International Conference on Software Engineering (ICSE 2008), Leipzig, Germany, May 10–18, 2008, pp. 261–270. doi:10.1145/1368088.1368124.

Gaia, F.N., Ferreira, G.C.S., Figueiredo, E., de Almeida Maia, M., 2014. A quantitative and qualitative assessment of aspectual feature modules for evolving software product lines. Sci. Comput. Program. 96, 230–253. doi:10.1016/j.scico.2014.03.006.

Gámez, N., Fuentes, L., 2011. Software product line evolution with cardinality-based feature models. In: Top Productivity through Software Reuse - 12th International Conference on Software Reuse, ICSR 2011, Pohang, South Korea, June 13–17, 2011. Proceedings, pp. 102–118. doi:10.1007/978-3-642-21347-2_9.

Gámez, N., Fuentes, L., 2013. Architectural evolution of famiware using cardinality-based feature models. Inf. Softw. Technol. 55 (3), 563–580. doi:10.1016/j.infsof.2012.06.012.

Ganesan, D., Lindvall, M., Ackermann, C., McComas, D., Bartholomew, M., 2009. Verifying architectural design rules of the flight software product line. In: Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24–28, 2009, Proceedings, pp. 161–170. doi:10.1145/1753235.1753258.

Garg, A., Critchlow, M., Chen, P., van der Westhuizen, C., van der Hoek, A., 2003. An environment for managing evolving product line architectures. In: 19th International Conference on Software Maintenance (ICSM 2003), The Architecture of Existing Systems, 22–26 September 2003, Amsterdam, The Netherlands, p. 358. doi:10.1109/ICSM.2003.1235443.

Greenfield, J., Short, K., 2003. Software factories: assembling applications with patterns, models, frameworks and tools. In: Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2003, October 26–30, 2003, Anaheim, CA, USA, pp. 16–27. doi:10.1145/949344.949348.

Gruschko, B., 2007. Changes classification in M2 models. In: Software Engineering 2007 - Beiträge zu den Workshops, Fachtagung des GI-Fachbereichs Softwaretechnik, 27.-30.3.2007 in Hamburg, pp. 277–280.

Guo, J., Wang, Y., Trinidad, P., Benavides, D., 2012. Consistency maintenance for evolving feature models. Expert Syst. Appl. 39 (5), 4987–4998. doi:10.1016/j.eswa.2011.10.014.

Heider, W., Froschauer, R., Grünbacher, P., Rabiser, R., Dhungana, D., 2010. Simulating evolution in model-based product line engineering. Inf. Softw. Technol. 52 (7), 758–769. doi:10.1016/j.infsof.2010.03.007.

Heider, W., Grünbacher, P., Rabiser, R., 2010. Negotiation constellations in reactive product line evolution. In: Fourth International Workshop on Software Product Management, IWSPM 2010, Sydney, NSW, Australia, September 27, 2010, pp. 63–66. doi:10.1109/IWSPM.2010.5623862.

Heider, W., Rabiser, R., Grünbacher, P., 2012. Facilitating the evolution of products in product line engineering by capturing and replaying configuration decisions. Int. J. Softw. Tools Technol. Transfer 14 (5), 613–630. doi:10.1007/s10009-012-0229-y.

Heider, W., Rabiser, R., Grünbacher, P., Lettner, D., 2012. Using regression testing to analyze the impact of changes to variability models on products. In: 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2–7, 2012, Volume 1, pp. 196–205. doi:10.1145/2362536.2362563.

Heider, W., Vierhauser, M., Lettner, D., Grünbacher, P., 2012. A case study on the evolution of a component-based product line. In: 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2012, Helsinki, Finland, August 20–24, 2012, pp. 1–10. doi:10.1109/WICSA-ECSA.212.8.

Hendrickson, S.A., van der Hoek, A., 2007. Modeling product line architectures through change sets and relationships. In: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20–26, 2007, pp. 189–198. doi:10.1109/ICSE.2007.56.

Heradio, R., Perez-Morago, H., Fernández-Amorós, D., Cabrerizo, F.J., Herrera-Viedma, E., 2016. A bibliometric analysis of 20 years of research on software product lines. Inf. Softw. Technol. 72, 1–15. doi:10.1016/j.infsof.2015.11.004.

Holdschick, H., 2012. Challenges in the evolution of model-based software product lines in the automotive domain. In: 4th International Workshop on Feature-Oriented Software Development, FOSD '12, Dresden, Germany - September 24, - 25, 2012, pp. 70–73. doi:10.1145/2377816.2377826.

Inoki, M., Kitagawa, T., Honiden, S., 2014. Application of requirements prioritization decision rules in software product line evolution. In: 5th IEEE International Workshop on Requirements Prioritization and Communication, RePriCo 2014, Karlskrona, Sweden, August 26, 2014, pp. 1–10. doi:10.1109/RePriCo.2014.6895216.

Jahn, M., Rabiser, R., Grünbacher, P., Löberbauer, M., Wolfinger, R., Mössenböck, H., 2012. Supporting model maintenance in component-based product lines. In: 2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, WICSA/ECSA 2012, Helsinki, Finland, August 20–24, 2012, pp. 21–30. doi:10.1109/WICSA-ECSA.212.10.

Jarzabek, S., Trung, H.D., 2011. Flexible generators for software reuse and evolution. In: Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21–28, 2011, pp. 920–923. doi:10.1145/1985793.1985946.

Jepsen, H.P., Beuche, D., 2009. Running a software product line: standing still is going backwards. In: Software Product Lines, 13th International Conference, SPLC 2009, San Francisco, California, USA, August 24–28, 2009, Proceedings, pp. 101–110. doi:10.1145/1753235.1753250.

Ji, W., Berger, T., Antkiewicz, M., Czarnecki, K., 2015. Maintaining feature traceability with embedded annotations. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 61–70. doi:10.1145/2791060.2791107.

Jiang, M., Zhang, J., Zhao, H., Zhou, Y., 2008. Maintaining software product lines - an industrial practice. In: 24th IEEE International Conference on Software Maintenance (ICSM 2008), September 28, - October 4, 2008, Beijing, China, pp. 444–447. doi:10.1109/ICSM.2008.4658100.

Johnson, R.E., Foote, B., 1988. Designing reusable classes. J.Object-Oriented Program. 1 (2), 22–35.

Júnior, R.J.S., Coelho, R., 2011. Preserving the exception handling design rules in software product line context: A practical approach. In: Dependable Computing Workshops (LADCW), 2011 Fifth Latin-American Symposium on, pp. 9–16. doi:10.1109/LADCW.2011.26.

Kakarontzas, G., Stamelos, I., Katsaros, P., 2008. Product line variability with elastic components and test-driven development. In: 2008 International Conferences on Computational Intelligence for Modelling, Control and Automation (CIMCA 2008), Intelligent Agents, Web Technologies and Internet Commerce (IAWTIC 2008), Innovation in Software Engineering (ISE 2008), 10–12 December 2008, Vienna, Austria, pp. 146–151. doi:10.1109/CIMCA.2008.84.

Kang, K.C., 1990. Feature-oriented Domain Analysis (FODA): Feasibility Study. Technical Report. Software Engineering Inst., Carnegie Mellon Univ.

Karimpour, R., Ruhe, G., 2013. Bi-criteria genetic search for adding new features into an existing product line. In: 1st International Workshop on Combining Modelling and Search-Based Software Engineering, CMSBSE at ICSE 2013, San Francisco, CA, USA, May 20, 2013, pp. 34–38. doi:10.1109/CMSBSE.2013.6604434.

Käßmeyer, M., Schulze, M., Schurius, M., 2015. A process to support a systematic change impact analysis of variability and safety in automotive functions. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 235–244. doi:10.1145/2791060.2791079.

Khurum, M., Gorschek, T., 2009. A systematic review of domain analysis solutions for product lines. J. Syst. Softw. 82 (12), 1982–2003. doi:10.1016/j.jss.2009.06.048.

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C.V., Loingtier, J., Irwin, J., 1997. Aspect-oriented programming. In: ECOOP, pp. 220–242. doi:10.1007/BFb0053381.

Kim, C.H.P., Czarnecki, K., 2005. Synchronizing cardinality-based feature models and their specializations. In: Model Driven Architecture - Foundations and Applications, First European Conference, ECMDA-FA 2005, Nuremberg, Germany, November 7–10, 2005, Proceedings, pp. 331–348. doi:10.1007/11581741_24.

Kitchenham, B.A., Charters, S., 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. Technical Report. Keele University and Durham University Joint Report.

Kitchenham, B.A., Travassos, G.H., von Mayrhauser, A., Niessink, F., Schneidewind, N.F., Singer, J., Takada, S., Vehvilainen, R., Yang, H., 1999. Towards an ontology of software maintenance. J. Softw. Maintenance 11 (6), 365–389. doi:10.1002/(SICI)1096-908X(199911/12)11:6⟨365::AID-SMR200⟩3.0.CO;2-W.

Knodel, J., Muthig, D., Naab, M., Lindvall, M., 2006. Static evaluation of software architectures. In: 10th European Conference on Software Maintenance and Reengineering (CSMR 2006), 22–24 March 2006, Bari, Italy, pp. 279–294. doi:10.1109/CSMR.2006.53.

Krishnan, S., Strasburg, C., Lutz, R.R., Goseva-Popstojanova, K., 2011. Are change metrics good predictors for an evolving software product line? In: Proceedings of the 7th International Conference on Predictive Models in Software Engineering, PROMISE 2011, Banff, Alberta, Canada, September 20–21, 2011, p. 7. doi:10.1145/2020390.2020397.

Krishnan, S., Strasburg, C., Lutz, R.R., Goseva-Popstojanova, K., Dorman, K.S., 2013. Predicting failure-proneness in an evolving software product line. Inf. Softw. Technol. 55 (8), 1479–1495. doi:10.1016/j.infsof.2012.11.008.

Krone, M., Snelting, G., 1994. On the inference of configuration structures from source code. In: Proceedings of the 16th International Conference on Software Engineering, Sorrento, Italy, May 16–21, 1994., pp. 49–57.

Krueger, C.W., 2001. Easing the transition to software mass customization. In: Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3–5, 2001, Revised Papers, pp. 282–293. doi:10.1007/3-540-47833-7_25.

Laguna, M.A., Crespo, Y., 2013. A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring. Sci. Comput. Program. 78 (8), 1010–1034. doi:10.1016/j.scico.2012.05.003.

Leopoldo Teixeira, P.B., Alves, V., Gheyi, R., 2015. A product line of theories for reasoning about safe evolution of product lines. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 161–170. doi:10.1145/2791060.2791105.

Lity, S., Lochau, M., Schaefer, I., Goltz, U., 2012. Delta-oriented model-based SPL regression testing. In: Proceedings of the Third International Workshop on Product LinE Approaches in Software Engineering, PLEASE 2012, Zurich, Switzerland, June 4, 2012, pp. 53–56. doi:10.1109/PLEASE.2012.6229772.

Liu, J., Dehlinger, J., Sun, H., Lutz, R.R., 2007. State-based modeling to support the evolution and maintenance of safety-critical software product lines. In: 14th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2007), 26–29 March 2007, Tucson, Arizona, USA, pp. 596–608. doi:10.1109/ECBS.2007.66.

Livengood, S., 2011. Issues in software product line evolution: complex changes in variability models. In: Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering, PLEASE 2011, Waikiki, Honolulu, HI, USA, May 22–23, 2011, pp. 6–9. doi:10.1145/1985484.1985487.

Lobato, L.L., Bittar, T.J., da Mota Silveira Neto, P.A., do Carmo Machado, I., de Almeida, E.S., de Lemos Meira, S.R., 2013. Risk management in software product line engineering: a mapping study. Int. J. Softw. Eng.Knowl. Eng. 23 (4), 523–558. doi:10.1142/S0218194013500150.

Loesch, F., Ploedereder, E., 2007. Restructuring variability in software product lines using concept analysis of product configurations. In: 11th European Conference on Software Maintenance and Reengineering, Software Evolution in Complex Software Intensive Systems, CSMR 2007, 21–23 March 2007, Amsterdam, The Netherlands, pp. 159–170. doi:10.1109/CSMR.2007.40.

Loughran, N., Rashid, A., 2004. Framed aspects: Supporting variability and configurability for AOP. In: Software Reuse: Methods, Techniques and Tools: 8th International Conference, ICSR 2004, Madrid, Spain, July 5–9, 2009. Proceedings, pp. 127–140. doi:10.1007/978-3-540-27799-6_11.

MacLean, A., Young, R.M., Bellotti, V., Moran, T.P., 1991. Questions, options, and criteria: elements of design space analysis. Hum. Comput. Interact. 6 (3–4), 201–250. doi:10.1080/07370024.1991.9667168.

Mcgregor, J.D., 2003. The Evolution of Product Line Assets. Technical Report. CMU/SEI-2003-TR-005, doi: 10.1109/MMUL.2003.1218250

McVoy, L., 2015. Preliminary product line support in bitkeeper. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 245–252. doi:10.1145/2791060.2791110.

Mende, T., Beckwermert, F., Koschke, R., Meier, G., 2008. Supporting the grow-and-prune model in software product lines evolution using clone detection. In: 12th European Conference on Software Maintenance and Reengineering, CSMR 2008, April 1–4, 2008, Athens, Greece, pp. 163–172. doi:10.1109/CSMR.2008.4493311.

Menkyna, R., Vranic, V., 2009. Aspect-oriented change realization based on multi-paradigm design with feature modeling. In: Advances in Software Engineering Techniques - 4th IFIP TC 2 Central and East European Conference on Software Engineering Techniques, CEE-SET 2009, Krakow, Poland, October 12–14, 2009. Revised Selected Papers, pp. 40–53. doi:10.1007/978-3-642-28038-2_4.

Merschen, D., Pott, J., Kowalewski, S., 2012. Integration and analysis of design artefacts in embedded software development. In: 36th Annual IEEE Computer Software and Applications Conference Workshops, COMPSAC 2012, Izmir, Turkey, July 16–20, 2012, pp. 503–508. doi:10.1109/COMPSACW.2012.94.

Michalik, B., Weyns, D., 2011. Towards a solution for change impact analysis of software product line products. In: 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011, Boulder, Colorado, USA, June 20–24, 2011, pp. 290–293. doi:10.1109/WICSA.2011.45.

Michalik, B., Weyns, D., Betsbrugge, W.V., 2011. On the problems with evolving egemin's software product line. In: Proceedings of the 2nd International Workshop on Product Line Approaches in Software Engineering, PLEASE 2011, Waikiki, Honolulu, HI, USA, May 22–23, 2011, pp. 15–19. doi:10.1145/1985484.1985489.

Montagud, S., Abrahão, S., Insfrán, E., 2012. A systematic review of quality attributes and measures for software product lines. Softw. Qual. J. 20 (3–4), 425–486. doi:10.1007/s11219-011-9146-7.

Montalvillo, L., Díaz, O., 2015. Tuning github for SPL development: branching models & repository operations for product engineers. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 111–120. doi:10.1145/2791060.2791083.

Moon, M., Chae, H.S., Nam, T., Yeom, K., 2007. A metamodeling approach to tracing variability between requirements and architecture in software product lines. In: Seventh International Conference on Computer and Information Technology (CIT 2007), October 16–19, 2007, University of Aizu, Fukushima, Japan, pp. 927–933. doi:10.1109/CIT.2007.117.

Murashkin, A., Antkiewicz, M., Rayside, D., Czarnecki, K., 2013. Visualization and exploration of optimal variants in product line engineering. In: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26, - 30, 2013, pp. 111–115. doi:10.1145/2491627.2491647.

Murthy, K., Kadur, A., Rao, P., 1994. A holistic approach to product marketability measurements-the pmm approach. In: Engineering Management Conference, 1994. 'Management in Transition: Engineering a Changing World', Proceedings of the 1994 IEEE International, pp. 323–329. doi:10.1109/IEMC.1994.379914.

Noor, M.A., Rabiser, R., Grünbacher, P., 2008. Agile product line planning: a collaborative approach and a case study. J. Syst. Softw. 81 (6), 868–882. doi:10.1016/j.jss.2007.10.028.

van Ommering, R.C., 2001. Roadmapping a product population architecture. In: Software Product-Family Engineering, 4th International Workshop, PFE 2001, Bilbao, Spain, October 3–5, 2001, Revised Papers, pp. 51–63. doi:10.1007/3-540-47833-7_6.

Padilha, J., Pereira, J.A., Figueiredo, E., Almeida, J.M., Garcia, A., Sant'Anna, C., 2014. On the effectiveness of concern metrics to detect code smells: an empirical study. In: Advanced Information Systems Engineering - 26th International Conference, CAiSE 2014, Thessaloniki, Greece, June 16–20, 2014. Proceedings, pp. 656–671. doi:10.1007/978-3-319-07881-6_44.

Paskevicius, P., Damasevicius, R., Stuikys, V., 2012. Change impact analysis of feature models. In: Information and Software Technologies - 18th International Conference, ICIST 2012, Kaunas, Lithuania, September 13–14, 2012. Proceedings, pp. 108–122. doi:10.1007/978-3-642-33308-8_10.

Passos, L.T., Guo, J., Teixeira, L., Czarnecki, K., Wasowski, A., Borba, P., 2013. Coevolution of variability models and related artifacts: a case study from the linux kernel. In: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26, - 30, 2013, pp. 91–100. doi:10.1145/2491627.2491628.

Pearse, T.T., Oman, P.W., 1997. Experiences developing and maintaining software in a multi-platform environment. In: ICSM, pp. 270–277. doi:10.1109/ICSM.1997.624254.

Peng, X., Yu, Y., Zhao, W., 2011. Analyzing evolution of variability in a software product line: From contexts and requirements to features. Inf. Softw. Technol. 53 (7), 707–721. doi:10.1016/j.infsof.2011.01.001.

Pereira, J.A., Constantino, K., Figueiredo, E., 2015. A systematic literature review of software product line management tools. In: Software Reuse for Dynamic Systems in the Cloud and Beyond - 14th International Conference on Software Reuse, ICSR 2015, Miami, FL, USA, January 4–6, 2015. Proceedings, pp. 73–89. doi:10.1007/978-3-319-14130-5_6.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008, University of Bari, Italy, 26–27 June 2008.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008b. Systematic mapping studies in software engineering.

Petersen, K., Vakkalanka, S., Kuzniarz, L., 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. Inf. Softw. Technol. 64, 1–18. doi:10.1016/j.infsof.2015.03.007.

Pichler, C., Huemer, C., Strommer, M., 2011. Evolution patterns for business document models. In: Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22–26, 2011. Workshop Proceedings (Volume 2), p. 21. doi:10.1145/2019136.2019160.

Planning Game agile practice. Http://c2.com/cgi/wiki?PlanningGame. Last visited: 2015-12-11.

Pleuss, A., Botterweck, G., Dhungana, D., Polzer, A., Kowalewski, S., 2012. Model-driven support for product line evolution on feature level. J. Syst. Softw. 85 (10), 2261–2274. doi:10.1016/j.jss.2011.08.008.

Pohl, K., Böckle, G., van der Linden, F., 2005. Software Product Line Engineering - Foundations, Principles, and Techniques. Springer doi:10.1007/3-540-28901-1.

Prehofer, C., 1997. Feature-oriented programming: a fresh look at objects. In: ECOOP, pp. 419–443. doi:10.1007/BFb0053389.

Quinton, C., Pleuss, A., Berre, D.L., Duchien, L., Botterweck, G., 2014. Consistency checking for the evolution of cardinality-based feature models. In: 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15–19, 2014, pp. 122–131. doi:10.1145/2648511.2648524.

Rabiser, R., Dhungana, D., Grünbacher, P., Lehner, K., Federspiel, C., 2007. Involving non-technicians in product derivation and requirements engineering: A tool suite for product line engineering. In: 15th IEEE International Requirements Engineering Conference, RE 2007, October 15-19th, 2007, New Delhi, India, pp. 367–368. doi:10.1109/RE.2007.26.

Ribeiro, M., Borba, P., 2008. Recommending refactorings when restructuring variabilities in software product lines. In: Second ACM Workshop on Refactoring Tools, WRT 2008, in conjunction with OOPSLA 2008, Nashville, TN, USA, October 19, 2008, p. 8. doi:10.1145/1636642.1636650.

Ribeiro, M., Borba, P., Kästner, C., 2014. Feature maintenance with emergent interfaces. In: 36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31, - June 07, 2014, pp. 989–1000. doi:10.1145/2568225.2568289.

Riva, C., Rosso, C. D., 2003. Experiences with software product family evolution, 161–169.

Romero, D., Urli, S., Quinton, C., Blay-Fornarino, M., Collet, P., Duchien, L., Mosser, S., 2013. SPLEMMA: a generic framework for controlled-evolution of software product lines. In: 17th International Software Product Line Conference co-located workshops, SPLC 2013 workshops, Tokyo, Japan - August 26 - 30, 2013, pp. 59–66. doi:10.1145/2499777.2500709.

Rubin, J., Czarnecki, K., Chechik, M., 2013. Managing cloned variants: a framework and experience. In: 17th International Software Product Line Conference, SPLC 2013, Tokyo, Japan - August 26, - 30, 2013, pp. 101–110. doi:10.1145/2491627.2491644.

Rubin, J., Czarnecki, K., Chechik, M., 2015. Cloned product variants: from ad-hoc to managed software product lines. STTT 17 (5), 627–646. doi:10.1007/s10009-014-0347-9.

Rubin, J., Kirshin, A., Botterweck, G., Chechik, M., 2012. Managing forked product variants. In: 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2–7, 2012, Volume 1, pp. 156–160. doi:10.1145/2362536.2362558.

Rumpe, B., Schulze, C., von Wenckstern, M., Ringert, J.O., Manhart, P., 2015. Behavioral compatibility of simulink models for product line maintenance and evolution. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 141–150. doi:10.1145/2791060.2791077.

Sabouri, H., Khosravi, R., 2011. Efficient verification of evolving software product lines. In: Fundamentals of Software Engineering - 4th IPM International Conference, FSEN 2011, Tehran, Iran, April 20–22, 2011, Revised Selected Papers, pp. 351–358. doi:10.1007/978-3-642-29320-7_24.

Sabouri, H., Khosravi, R., 2014. Reducing the verification cost of evolving product families using static analysis techniques. Sci. Comput. Program. 83, 35–55. doi:10.1016/j.scico.2013.06.009.

Santos, A.R., de Oliveira, R.P., de Almeida, E.S., 2015. Strategies for consistency checking on software product lines: a mapping study. In: Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering, EASE 2015, Nanjing, China, April 27–29, 2015, pp. 5:1–5:14. doi:10.1145/2745802.2745806.

Sarang, N., Sanglikar, M.A., 2007. An analysis of effort variance in software maintenance projects. In: Advances in Computer and Information Sciences and Engineering, Proceedings of the 2007 International Conference on Systems, Computing Sciences and Software Engineering (SCSS), part of the International Joint Conferences on Computer, Information, and Systems Sciences, and Engineering (CISSE 2007), Bridgeport, CT, USA, December 3–12, 2007, pp. 366–371. doi:10.1007/978-1-4020-8741-7_66.

Savolainen, J., 2013. Past, present and future of product line engineering in industry: reflecting on 15 years of variability management in real projects. In: Proceedings of the Eighth International Workshop on Variability Modelling of Software-Intensive Systems. ACM, New York, NY, USA, pp. 1:1–1:1. doi:10.1145/2556624.2557789.

Savolainen, J., Kuusela, J., 2001. Violatility analysis framework for product lines, 133–141. 10.1145/375212.375277

Savolainen, J., Kuusela, J., 2008. Scheduling product line features for effective roadmapping. In: 15th Asia-Pacific Software Engineering Conference (APSEC 2008), 3–5 December 2008, Beijing, China, pp. 195–202. doi:10.1109/APSEC.2008.21.

Schackmann, H., Lichter, H., 2006. A cost-based approach to software product line management. In: International Workshop on Software Product Management, IWSPM '06, Minneapolis/St.Paul, Minnesota, USA, September 12, 2006, pp. 13–18. doi:10.1109/IWSPM.2006.1.

Schaefer, I., Bettini, L., Bono, V., Damiani, F., Tanzarella, N., 2010. Delta-oriented programming of software product lines. In: Software Product Lines: Going Beyond - 14th International Conference, SPLC 2010, Jeju Island, South Korea, September 13–17, 2010. Proceedings, pp. 77–91. doi:10.1007/978-3-642-15579-6_6.

Scheidemann, K.D., 2006. Optimizing the selection of representative configurations in verification of evolving product lines of distributed embedded systems. In: Software Product Lines, 10th International Conference, SPLC 2006, Baltimore, Maryland, USA, August 21–24, 2006, Proceedings, pp. 75–84. doi:10.1109/SPLINE.2006.1691579.

Schmid, K., Eichelberger, H., 2007. A requirements-based taxonomy of software product line evolution. ECEASST 8, 1–13.

Schmid, K., Rabiser, R., Grünbacher, P., 2011. A comparison of decision modeling approaches in product lines. In: Fifth International Workshop on Variability Modelling of Software-Intensive Systems, Namur, Belgium, January 27–29, 2011. Proceedings, pp. 119–126. doi:10.1145/1944892.1944907.

Schmid, K., Verlage, M., 2002. The economic impact of product line adoption and evolution. IEEE Softw. 19 (4), 50–57. doi:10.1109/MS.2002.1020287.

Schröter, R., Siegmund, N., Thüm, T., Saake, G., 2014. Feature-context interfaces: tailored programming interfaces for software product lines. In: 18th International Software Product Line Conference, SPLC '14, Florence, Italy, September 15–19, 2014, pp. 102–111. doi:10.1145/2648511.2648522.

Schubanz, M., Pleuss, A., Pradhan, L., Botterweck, G., Thurimella, A. K., 2013. Model-driven planning and monitoring of long-term software product line evolution, 18:1–18:5. 10.1145/2430502.2430527

Schulze, S., Lochau, M., Brunswig, S., 2013. Implementing refactorings for FOP: lessons learned and challenges ahead. In: 5th International Workshop on Feature-Oriented Software Development, FOSD '13, Indianapolis, IN, USA, October 26, 2013, pp. 33–40. doi:10.1145/2528265.2528271.

Schulze, S., Thüm, T., Kuhlemann, M., Saake, G., 2012. Variant-preserving refactoring in feature-oriented software product lines. In: Sixth International Workshop on Variability Modelling of Software-Intensive Systems, Leipzig, Germany, January 25–27, 2012. Proceedings, pp. 73–81. doi:10.1145/2110147.2110156.

SEBOK maintainability Http://sebokwiki.org/wiki/Reliability,_Availability,_and_Maintainability Last visited: 2015-12-11.

Seidl, C., Heidenreich, F., Aßmann, U., 2012. Co-evolution of models and feature mapping in software product lines. In: 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2–7, 2012, Volume 1, pp. 76–85. doi:10.1145/2362536.2362550.

Sharp, D.C., 1999. Exploiting object technology to support product variability. In: Proceedings of the 18th Digital Avionics Systems Conference, 2 doi:10.1109/DASC.1999.863671. 9.C.1–1–9.C.1–8

Shen, L., Peng, X., Zhao, W., 2009. A comprehensive feature-oriented traceability model for software product line development. In: 20th Australian Software Engineering Conference (ASWEC 2009), 14–17 April 2009, Gold Cost, Australia, pp. 210–219. doi:10.1109/ASWEC.2009.27.

Shen, L., Peng, X., Zhu, J., Zhao, W., 2010. Synchronized architecture evolution in software product line using bidirectional transformation. In: Proceedings of the 34th Annual IEEE International Computer Software and Applications Conference, COMPSAC 2010, Seoul, Korea, 19–23 July 2010, pp. 389–394. doi:10.1109/COMPSAC.2010.71.

Singer, J., 1998. Practices of software maintenance. In: 1998 International Conference on Software Maintenance, ICSM 1998, Bethesda, Maryland, USA, November 16–19, 1998, pp. 139–145. doi:10.1109/ICSM.1998.738502.

Svahnberg, M., Bosch, J., 1999. Evolution in software product lines: two cases. J. Softw. Maintenance 11 (6), 391–422. doi:10.1002/(SICI)1096-908X(199911/12)11:6⟨391::AID-SMR199⟩3.0.CO;2-8.

Svahnberg, M., Bosch, J., 2000. Issues concerning variability in software product lines. In: Software Architectures for Product Families, International Workshop IW-SAPF-3, Las Palmas de Gran Canaria, Spain, March 15–17, 2000, Proceedings, pp. 146–157. doi:10.1007/978-3-540-44542-5_17.

Swanson, E.B., 1976. The dimensions of maintenance. In: Proceedings of the 2nd International Conference on Software Engineering, San Francisco, California, USA, October 13–15, 1976., pp. 492–497.

Taborda, L.J.M., 2004. Generalized release planning for product line architectures. In: Software Product Lines, Third International Conference, SPLC 2004, Boston, MA, USA, August 30-September 2, 2004, Proceedings, pp. 238–254. doi:10.1007/978-3-540-28630-1_15.

Tartler, R., Sincero, J., Schröder-Preikschat, W., Lohmann, D., 2009. Dead or alive: finding zombie features in the linux kernel. In: Proceedings of the First International Workshop on Feature-Oriented Software Development, FOSD 2009, Denver, Colorado, USA, October 6, 2009, pp. 81–86. doi:10.1145/1629716.1629732.

Teixeira, L., Borba, P., Gheyi, R., 2015. Safe evolution of product populations and multi product lines. In: Proceedings of the 19th International Conference on Software Product Line, SPLC 2015, Nashville, TN, USA, July 20–24, 2015, pp. 171–175. doi:10.1145/2791060.2791084.

ter Beek, M.H., Muccini, H., Pelliccione, P., 2011. Guaranteeing correct evolution of software product lines: Setting up the problem. In: Software Engineering for Resilient Systems - Third International Workshop, SERENE 2011, Geneva, Switzerland, September 29–30, 2011. Proceedings, pp. 100–105. doi:10.1007/978-3-642-24124-6_9.

ter Beek, M.H., Muccini, H., Pelliccione, P., 2012. Assume-guarantee testing of evolving software product line architectures. In: Software Engineering for Resilient Systems - 4th International Workshop, SERENE 2012, Pisa, Italy, September 27–28, 2012. Proceedings, pp. 91–105. doi:10.1007/978-3-642-33176-3_7.

Tesanovic, A., 2007. Evolving embedded product lines: opportunities for aspects. In: Proceedings of the 6th workshop on Aspects, Components, and Patterns for Infrastructure Software, ACP4IS 2007, Vancouver, British Columbia, Canada, March 12, 2007, p. 10. doi:10.1145/1233901.1233911.

Thao, C., Munson, E.V., Nguyen, T.N., 2008. Software configuration management for product derivation in software product families. In: 15th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems (ECBS 2008), 31 March - 4 April 2008, Belfast, Northern Ireland, pp. 265–274. doi:10.1109/ECBS.2008.53.

Thüm, T., Apel, S., Kästner, C., Schaefer, I., Saake, G., 2014. A classification and survey of analysis strategies for software product lines. ACM Comput. Surv. 47 (1), 6:1–6:45. doi:10.1145/2580950.

Thüm, T., Batory, D.S., Kästner, C., 2009. Reasoning about edits to feature models. In: 31st International Conference on Software Engineering, ICSE 2009, May 16–24, 2009, Vancouver, Canada, Proceedings, pp. 254–264. doi:10.1109/ICSE.2009.5070526.

Thurimella, A.K., Bruegge, B., 2007. Evolution in product line requirements engineering: a rationale management approach. In: 15th IEEE International Requirements Engineering Conference, RE 2007, October 15-19th, 2007, New Delhi, India, pp. 254–257. doi:10.1109/RE.2007.11.

Thurimella, A.K., Bruegge, B., 2012. Issue-based variability management. Inf. Softw. Technol. 54 (9), 933–950. doi:10.1016/j.infsof.2012.02.005.

Thurimella, A.K., Bruegge, B., Creighton, O., 2008. Identifying and exploiting the similarities between rationale management and variability management. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8–12, 2008, Proceedings, pp. 99–108. doi:10.1109/SPLC.2008.14.

Thurimella, A.K., Brügge, B., 2013. A mixed-method approach for the empirical evaluation of the issue-based variability modeling. J. Syst. Softw. 86 (7), 1831–1849. doi:10.1016/j.jss.2013.01.038.

Tischer, C., Boss, B., Müller, A., Thums, A., Acharya, R., Schmid, K., 2012. Developing long-term stable product line architectures. In: 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2–7, 2012, Volume 1, pp. 86–95. doi:10.1145/2362536.2362551.

Tizzei, L.P., Dias, M.O., Rubira, C.M.F., Garcia, A., Lee, J., 2011. Components meet aspects: Assessing design stability of a software product line. Information & Software Technology 53 (2), 121–136. doi:10.1016/j.infsof.2010.08.007.

Tofan, D., Galster, M., Avgeriou, P., Schuitema, W., 2014. Past and future of software architectural decisions - A systematic mapping study. Inf. Softw. Technol. 56 (8), 850–872. doi:10.1016/j.infsof.2014.03.009.

Tran, L.M.S., Massacci, F., 2014. An approach for decision support on the uncertainty in feature model evolution. In: IEEE 22nd International Requirements Engineering Conference, RE 2014, Karlskrona, Sweden, August 25–29, 2014, pp. 93–102. doi:10.1109/RE.2014.6912251.

Vale, G., Figueiredo, E., Abílio, R., Costa, H.A.X., 2014. Bad smells in software product lines: A systematic review. In: Eighth Brazilian Symposium on Software Components, Architectures and Reuse, SBCARS 2014, Maceió, Alagoas, Brazil, September 29–30, 2014, pp. 84–94. doi:10.1109/SBCARS.2014.21.

van der Linden, F., Schmid, K., Rommes, E., 2007. Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. Springer doi:10.1007/978-3-540-71437-8.

van Gurp, J., Bosch, J., 2002. Design erosion: problems and causes. J. Syst. Softw. 61 (2), 105–119. doi:10.1016/S0164-1212(01)00152-2.

Vianna, A., Pinto, F., Sena, D., Kulesza, U., Coelho, R., Santos, J., Lima, J., Lima, G., 2012. Squid: an extensible infrastructure for analyzing software product line implementations. In: 16th International Software Product Line Conference, SPLC '12, Salvador, Brazil - September 2–7, 2012, Volume 2, pp. 209–216. doi:10.1145/2364412.2364447.

Vierhauser, M., Grünbacher, P., Heider, W., Holl, G., Lettner, D., 2012. Applying a consistency checking framework for heterogeneous models and artifacts in industrial product lines. In: Model Driven Engineering Languages and Systems - 15th International Conference, MODELS 2012, Innsbruck, Austria, September 30-October 5, 2012. Proceedings, pp. 531–545. doi:10.1007/978-3-642-33666-9_34.

Vierhauser, M., Rabiser, R., Grünbacher, P., 2014. A requirements monitoring infrastructure for very-large-scale software systems. In: Requirements Engineering: Foundation for Software Quality - 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7–10, 2014. Proceedings, pp. 88–94. doi:10.1007/978-3-319-05843-6_7.

Villela, K., Dörr, J., John, I., 2010. Evaluation of a method for proactively managing the evolving scope of a software product line. In: Requirements Engineering: Foundation for Software Quality, 16th International Working Conference, REFSQ 2010, Essen, Germany, June 30, - July 2, 2010. Proceedings, pp. 113–127. doi:10.1007/978-3-642-14192-8_13.

Völter, M., Visser, E., 2011. Product line engineering using domain-specific languages. In: Software Product Lines - 15th International Conference, SPLC 2011, Munich, Germany, August 22–26, 2011, pp. 70–79. doi:10.1109/SPLC.2011.25.

Walrad, C.C., Strom, D., 2002. The importance of branching models in SCM. IEEE Comput. 35 (9), 31–38. doi:10.1109/MC.2002.1033025.

Weiss, D.M., 2008. The product line hall of fame. In: Software Product Lines, 12th International Conference, SPLC 2008, Limerick, Ireland, September 8–12, 2008, Proceedings, p. 395. doi:10.1109/SPLC.2008.56.

Weyns, D., Michalik, B., Helleboogh, A., Boucké, N., 2011. An architectural approach to support online updates of software product lines. In: 9th Working IEEE/IFIP Conference on Software Architecture, WICSA 2011, Boulder, Colorado, USA, June 20–24, 2011, pp. 204–213. doi:10.1109/WICSA.2011.34.

Wieringa, R., Maiden, N.A.M., Mead, N.R., Rolland, C., 2006. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. Requir. Eng. 11 (1), 102–107. doi:10.1007/s00766-005-0021-6.

Wohlin, C., Runeson, P., da Mota Silveira Neto, P.A., EngstrÃ¶m, E., do Carmo Machado, I., de Almeida, E.S., 2013. On the reliability of mapping studies in software engineering. J. Syst. Softw. 86 (10), 2594–2610. doi:10.1016/j.jss.2013.04.076.

Yau, S.S., Collofello, J.S., MacGregor, T.M., 1978. Ripple effect analysis of software maintenance. In: International Computer Software and Applications Conference, pp. 71–82.

Yazdanshenas, A.R., Moonen, L., 2012. Fine-grained change impact analysis for component-based product families. In: 28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23–28, 2012, pp. 119–128. doi:10.1109/ICSM.2012.6405262.

Yu, D., Geng, P., Wu, W., 2012. Constructing traceability between features and requirements for software product line engineering. In: 19th Asia-Pacific Software Engineering Conference - Workshops, APSEC 2012, Hong Kong, China, December 4–7, 2012, pp. 27–34. doi:10.1109/APSEC.2012.135.

Leticia Montalvillo is a PhD student at the University of the Basque Country (UPV/EHU). Her current research interests include Software Product Lines and Configuration Management. Montalvillo obtained a BSc from the University of Mondragon and a MSc in Information Technology from the Polytechnic University of Catalonia (BarcelonaTech). Contact her at leticia.montalvillo@ehu.eus.

Oscar Díaz is Full Professor at the University of the Basque Country (UPV/EHU). His current interests include Web2.0, model-driven engineering and Software Product Lines. He leads a fifteen-member group, ONEKIN, with a focus on Web Engineering and close partnership with industry. He obtained the BSc in Computing at the University of the Basque Country, and a PhD by the University of Aberdeen. Contact him at Facultad de Informática, Apdo. 649, 20.011 San Sebastián (Spain), oscar.diaz@ehu.eus.