

How to Retire and Replace a Software Product Line

Alejandro Cortiñas*

Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
alejandro.cortinas@udc.es

Jacob Krüger

Eindhoven University of Technology
Eindhoven, The Netherlands
j.kruger@tue.nl

Victor Lamas

Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
victor.lamas@udc.es

Miguel R. Luaces

Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
miguel.luaces@udc.es

Oscar Pedreira

Universidade da Coruña, CITIC
Research Center, Database Lab.
A Coruña, Spain
opedreira@udc.es

ABSTRACT

Software product-line engineering provides a framework for an organization to develop a family of similar, yet customized, software systems based on a common platform. This platform allows the organization to configure a system to changing customer requirements, while also achieving long-term benefits like reduced development and maintenance costs. Consequently, a product line is typically used for a long-living family of systems and is continuously evolved. However, at some point even a product line may be retired and potentially replaced by a successor, for instance, because of outdated technology that cannot be replaced easily and thus makes developing a new product line more feasible. Such a retiring of product lines is mentioned in previous work, but has not been investigated in detail. In this paper, we tackle this gap by describing a process for retiring and replacing a product line, which we defined based on a real-world action-research-like case study. Via this case study, we describe how our process can be executed in practice, what decisions must be considered, as well as the pros and cons we experienced with retiring a product line. We expect that these contributions will help practitioners retire product lines more systematically and with fewer problems. We also indicate open research directions that should be tackled in the future.

CCS CONCEPTS

• **Software and its engineering** → **Software product lines**; **Reusability**; *Abstraction, modeling and modularity*; *Software architectures*.

*All authors have contributed equally and the names are listed in alphabetical order.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SPLC'23, August 28 – September 1, 2023, Tokyo, Japan

© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/10.1145/3579027.3609004>

KEYWORDS

Software product line engineering, domain analysis, requirements analysis, product line modernization, case study, mobility information systems

ACM Reference Format:

Alejandro Cortiñas, Jacob Krüger, Victor Lamas, Miguel R. Luaces, and Oscar Pedreira. 2023. How to Retire and Replace a Software Product Line. In *27th ACM International Systems and Software Product Line Conference - Volume A (SPLC '23)*, August 28–September 1, 2023, Tokyo, Japan. ACM, New York, NY, USA, 12 pages. <https://doi.org/10.1145/3579027.3609004>

1 INTRODUCTION

A software product line (SPL) is a software-engineering concept that allows for the efficient and systematic reuse of software artifacts across multiple related software systems [6, 35, 45]. In other words, an SPL is a group of systems that are built on a common platform sharing a similar architecture, design, and features, but that can also be customized to suit particular customer needs. With each new system variant, developers can reuse the platform's existing artifacts, such as source code, specifications, or models, rather than starting from scratch. For this purpose, the developers engineer a platform (i.e., domain engineering) by eliciting features for the SPL [26, 27], establishing a variability model [14, 42], designing the architecture [20, 43], implementing a variability mechanism [24, 60], setting up testing infrastructures [18, 21], and using a configurator tool [44, 54]. Through a valid feature configuration, a system variant of the SPL is customized to user requirements and automatically derived (i.e., application engineering). By building upon the common platform, developers can ensure that systems within the SPL share a consistent and standardized development methodology. Consequently, an SPL promises several benefits by enabling developers to reuse artifacts for creating systems more efficiently, such as reduced development and maintenance costs, faster time-to-market, and improved software quality [28, 33, 50, 57].

SPLs have been used in various domains, including aerospace, electronic equipment, instruments and components, automotive, embedded firmware, software tools, or healthcare equipment and supplies [7, 23, 33, 34, 36, 39, 55, 57]. Typically, an SPL represents a set of long-living systems, and thus is subject to changes for various reasons, such as changes in requirements, the platform architecture, or the technology used. Managing the evolution of SPL artifacts is

more challenging than for traditional systems, due to the variability and consequent complexity these artifacts exhibit. Various means have been proposed for managing the evolution of SPL artifacts, including model-based techniques, catalogs of evolution operators and patterns, or code versioning [5, 9, 11, 17, 37, 40, 48].

However, at some point in time, the developers or organization may deem it necessary to deprecate an SPL and to create a new one. For example, an SPL may become technologically outdated, decline in quality, may exhibit a too complex architecture, or may no longer meet the needs of the market. In such cases, it may be more cost-effective to start fresh with a new SPL that incorporates the latest technologies and best practices. Unfortunately, while a few publications mention that legacy SPLs are replaced or retired [10, 22, 39, 53], there have been no detailed studies of this evolution scenario. As a consequence, we are lacking knowledge regarding the causes (e.g., outdated technology), decisions (e.g., start from scratch or what to reuse from the legacy SPL), and consequences (e.g., problems) that relate to retiring an SPL.

In this paper, we present a process for retiring an SPL and replacing it with a new one, which we employed in an industrial case study. We describe not only the process, but also what decisions were important, what challenges we experienced, as well as how to balance retiring and reusing the legacy SPL. Essentially, this process connects two instances of (domain) engineering an SPL based on typical processes, such as domain and application engineering [6, 45] or promote-pl [35]. With our action-research-like case study, we provide an in-depth analysis of a real-world scenario in which we employed the process, contributing valuable insights into this specific SPL evolution scenario.

In more detail, we contribute the following:

- We present a process for retiring an SPL, including relevant decisions and their consequences.
- We report our experiences of executing this process in an industrial case study of replacing an SPL with a new one.
- We discuss the challenges we experienced and their implications for practitioners as well as researchers.

Our contributions provide a detailed understanding of how SPLs can be retired and the associated practical as well as research challenges. These contributions help practitioners decide whether and how to retire an SPL, and researchers in understanding this specific process with its corresponding challenges that require further research.

2 RELATED WORK

We are concerned with retiring and replacing one SPL with a new one, which is a highly specific evolution scenario. Following established process models for SPL engineering [6, 35, 45], we could describe this scenario as the end-of-life of one SPL engineering cycle and the start of a new one. Consequently, this scenario is related to the extractive SPL adoption strategy [30], in which existing legacy systems are reused to develop an SPL. However, different meta-studies on SPL extraction and evolution [31, 35, 37, 40, 46, 52] highlight that the focus in this area of research is on adopting an SPL from a set of cloned software systems (i.e., not from an SPL). The research on SPL evolution focuses on one existing SPL and its continuous modernization. Similarly, research on scoping and maturity assessments in the context of SPL adoption [1, 2, 8, 9, 29,

38, 39, 41, 47, 49, 56] is also related to our work, since it focuses on how to decide whether and to what extent to adopt an SPL.

Despite such extensive research, we are not aware of a study that describes our specific scenario in detail. In this scenario, an organization faces very different decisions than in the typical extractive strategy, for instance, deciding which existing artifacts could be reused can be more challenging. Specifically, if an SPL is adopted from cloned systems, this is typically driven by increasing challenges of maintaining these systems [19, 23, 36, 59]—but it is clear that these and more similar systems are demanded by the market. Consequently, it is typically clear that the artifacts of the legacy systems should be reused, since they are up-to-date, demanded by customers, and established in the domain.

In contrast, in our scenario, there can be various reasons for retiring an SPL, such as outdated technology, an unfeasible architecture, or missing market demands. As a consequence, it is much less clear whether and how which artifacts of the existing SPL may be useful to reuse or should be developed from scratch, enforcing different decisions onto organizations and developers. Even though this scenario has been mentioned in some related research, the processes, decisions, and their implications have not been investigated in detail. Arguably the closest work on this scenario is the case study by Svahnberg and Mattsson [53], who report on migrating from one SPL generation to a new one. However, this work is more than 20 years old, describes only the migration planning (i.e., architecture design), stays on a very abstract level, and reports on moving from a hardware-centered product line into a full-fledged SPL. As such, the actual process of moving from one SPL to a new one is unclear and our work provides much richer details. Other studies report that legacy SPLs exist that are, or already have been, retired and replaced by new ones [10, 22, 39]. However, none of these works focuses on the details of this particular evolution scenario. Overall, our contributions complement the existing research by providing insights into a specific evolution scenario that is relevant in practice, but has not been systematically studied in research.

3 STUDY SETTING AND METHODOLOGY

In the following, we first report the context of our research before defining our research questions and methodology.

3.1 Context of our Research

New technologies for collecting mobility data of public transportation allow administrations to better understand mobility patterns, such as traffic jams or empty rides. Analyzing such vast datasets is highly advantageous as it facilitates the development of new services and applications, such as apps that display the real-time location of buses or traffic conditions. Additionally, these analyses are crucial for governments and administrations when planning new transport infrastructures, and for transportation companies seeking insights into their customers' behavior. Furthermore, analyzing this data provides a better understanding of population distribution challenges, particularly the phenomena of low-density areas and depopulation. By combining mobility data with demographic information (such as population size and characteristics), a variety of applications can be developed to improve public policies. For

instance, this information can be used to identify the optimal areas for establishing new day care centers, schools, or nursing homes.

Analyzing citizens' mobility patterns is not a new concept. Historically, city transportation officials conducted surveys to gather mobility data, such as asking subway riders which stops they used and which buses they took before and after entering the subway. This information was then utilized to generate an origin-destination matrix, which depicted the number of individuals traveling from one location to another throughout the city during specific hours on working and non-working days. The significant change today is that technology allows to automatically capture travelers' entrance into the transportation network via transport cards and ticket reading devices, which simplifies tracing their travel patterns.

Another increasingly important topic in urban mobility management are low-emission zones. They are becoming a widespread trend across Europe, as cities seek to reduce air pollution and promote sustainable transportation. In fact, as of 2022, more than 300 cities in Europe have implemented some form of low-emission zone, according to the CLARS (Charging, Low emission zones, other Access Regulation Schemes) website.¹ As an example, a law requiring the creation and implementation of a mobility plan that includes the management of low-emission zones has been adopted in Spain and applies to cities with a population over 50,000 inhabitants (approximately 150 cities). The trend is undeniable: More and more cities in Europe are realizing the necessity to combat air pollution and advance sustainable mobility, even though the specifics of low-emission zones vary depending on the city and country. Given that the European Union is establishing increasingly ambitious goals for decreasing emissions and encouraging sustainable mobility, this tendency will likely only get stronger over the next years.

Given the increasing importance of mobility data analysis, we believe that it is highly relevant to develop an SPL for a product family of mobility information systems. Thus, we (i.e., the authors from Universidade da Coruña) submitted and received funding for a project proposal to the Spanish National Research Agency aimed at developing a platform called PLAGEMIS (PLATform for the GEneration of Mobility Information Systems) that will automate the generation of Mobility Information Systems (MISs) using variability management techniques, such as SPLs, domain specific languages, and architecture scaffolding. In addition, our research group in Spain has a spin-off company and we believe that the SPL has a great potential to be transferred to more industry partners.

During a previous industry-research collaboration [4, 12, 13, 15], we explored the development of Geographic Information Systems (GISs) using SPLs. GISs are characterized by their ability to manage geospatial entities, which has an impact on all levels of software, including the database (which must support geospatial data types and operations) and the user interface (which always displays information in maps consisting of layers). In addition to these general functionalities, GISs provide specific features like route calculation and data processing based on spatial representation. Public administrations and private companies frequently use GISs to manage infrastructures (e.g., transportation or supply networks) or mobility scenarios (e.g., logistics or mobile workforces). Organizations like

the International Organization for Standardization (ISO)² or Open Geospatial Consortium (OGC)³ have standardized a GIS software architecture and its components, resulting in most GISs having the same software architecture and sharing tools, libraries, as well as software components in their development, regardless of their concrete application domain. Consequently, many functionalities are shared among GISs in different domains. This was the motivation to explore the application of an SPL to GISs: our final GIS SPL comprises the features that may appear in any GIS and allows a software engineer to select which of them must be included in a specific product variant.

In our spin-off company, we have the same vision for PLAGEMIS and, in fact, the new MIS SPL shall replace the GIS SPL. Consequently, we can build on our previous experiences with developing the MIS SPL, but we also needed to decide what parts of the old SPL we had to retire or could reuse. Within this paper, we report our experiences regarding such decisions.

3.2 Methodology

While the GIS SPL includes some of the functionalities required for the new MIS SPL, we believe that some of its parts are outdated in terms of technology, that it includes irrelevant functionality, and that creating a new SPL would require less effort than adapting the existing one. After a literature review, we did not find detailed studies of the scenario where an SPL is deprecated and a new one is created from its assets. Consequently, we decided to investigate this scenario in more detail. To select a research method, we followed the guidelines by Wohlin and Runeson [58]. Given that the main goal of our research was replacing an SPL, and that applying a technology transfer model does not seem feasible to obtain reliable scientific insights, we decided to follow an action-research-like methodology as described by Staron [51]. We split our group of authors into two to apply the action research method. The first three authors took the position of researchers, because they are more involved with the funded research project and/or have a more objective perspective on the research questions. In contrast, the last two authors took the role of the industry stakeholders, since they are closely connected to the spin-off company, and thus the industrial perspective. This division gave us the chance to learn from both points of view and make sure the methodology we came up with was useful to both, researchers and practitioners. We display an overview of the action-research methodology in Figure 1, according to which we structure the remainder of this paper.

During the first diagnosis step of the action-research methodology, we analyzed the current state of our GIS SPL and its feasibility for the new MIS SPL. This step led us to the conclusion that the SPL may not be appropriate for the project and has to be deprecated. However, we were unable to identify a technique that specifically deals with deprecating an existing SPL and building on its assets to construct a new one. We believe that it is important to provide a systematic process for this task and to define criteria for determining whether to reuse legacy SPL artifacts. Therefore, the research questions (RQs) that we aimed to answer are:

RQ₁ How can we retire an SPL and replace it with a new one?

¹<https://urbanaccessregulations.eu/>

²<https://iso.org>

³<https://www.ogc.org>

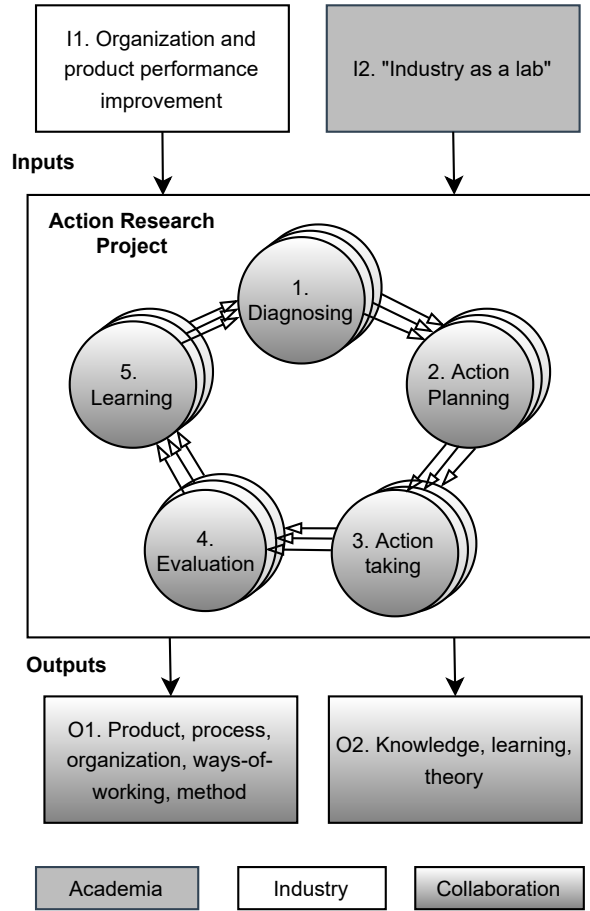


Figure 1: Overview of the action research methodology according to Wohlin and Runeson [58].

RQ₂ What are the challenges of retiring and replacing an SPL?

RQ₃ How can we decide whether to reuse legacy SPL artifacts?

By answering these research questions, we hope to contribute novel insights that can help other practitioners in similar cases, and guide researchers towards new directions.

Our action research methodology involved four major iterations (cf. Figure 1). During the first iteration, our planned actions aimed to analyze the requirements and specifications of the new MIS SPL. This analysis helped us develop a clear understanding of what the new SPL needed to be and to set clear goals for the rest of the project. During the second iteration, we focused on analyzing the existing GIS SPL to determine whether it could be reused for the new project or whether it needed to be deprecated. As a result of this analysis, we determined that the existing SPL was not suitable for the new project and needed to be deprecated. In the third iteration, our actions were oriented towards merging the assets of the existing with those of the new SPL. Finally, during the fourth and final iteration, our actions were targeted at defining a methodology for deprecating the existing SPL and creating the new one based on its assets. We conducted collaborative meetings at the conclusion of each iteration to assess the outcomes and consolidate

Table 1: Initial ISO 14813-1 features we selected.

ISO Service Domain	Included	Features in Model
Domain A: Traveller information	Partially	37
Domain B: Traffic Management and Operations	Partially	10
Domain C: Vehicle Services	No	
Domain D: Freight Transport	No	
Domain E: Public Transport	Partially	15
Domain F: Emergency Service	No	
Domain G: Transport related payment	No	
Domain H: Road transport relate personal safety	No	
Domain I: Weather and environmental conditions monitoring	Partially	3
Domain J: Disaster response management and coordination	No	
Domain K: National security	No	
Domain L: ITS data management	No	
Domain M: Performance management	No	

the lessons learned. Specifically, we formalized the findings of our investigation during these discussions and pinpointed areas in need of improvement. We recorded our results to ensure that we could answer our research questions and contribute reliable insights for others to build upon.

4 CONDUCT

In this section, we describe how we applied the action-research methodology that we defined in Section 3.1. We first describe our SPL definition process in Section 4.1. Then, we present our process of reviewing the existing SPL architecture in Section 4.2. After that, we describe our process of implementing the new SPL in Section 4.3. Finally, we describe the methodology that we have defined from the lessons learned in Section 4.4.

4.1 First Iteration: Definition of the SPL

During the first iteration of the action research, our focus was on analyzing the requirements and specifications for the new SPL. As we describe in Section 3.1, our research group has been granted funding by the Spanish National Research Agency to develop a platform called PLAGEMIS to automate the generation of MISs using an SPL. To carry out the domain analysis of the PLAGEMIS SPL, we extracted and recorded all requirements necessary to meet the project objectives from the project proposal of PLAGEMIS.

Once we elicited the set of requirements for the SPL, we analyzed reference architectures to identify the essential abstractions and notions that will serve as the foundation of the new SPL. PLAGEMIS aims to integrate the features of MISs, which are a subset of Intelligent Transport Systems (ITSs). ITSs are a wide-ranging concept that encompasses various technologies designed to enhance the effectiveness, reliability, and sustainability of transportation systems. Such systems have garnered considerable attention in academic literature and are described in a set of ISO standards, including ISO 14813-1 [25]. This standard focuses on the functional architecture of ITSs and provides guidelines for designing and deploying ITSs. It follows a top-down methodology to create a hierarchy of *service*

Table 2: Overview of the feature model evolution.

ISO Feature Model		Dashboard Refined		Experts Refined	
Additions	65	Additions	24	Additions	6
		Deletions	31		
		Changes	27		

domains, *service groups*, and *services* to be used as the input for ITS architectures. After undertaking a thorough analysis of the ISO 14813-1 standard, we selected a subset of services that must be included in the PLAGEMIS SPL. Among the services we selected were real-time traffic updates, car tracking, and interaction with public transportation networks. We constructed a feature model for our new SPL using the descriptions provided by ISO 14813-1 for each of these services. In Table 1 we summarize the result of this process, enumerating for each ISO service domain whether we included it in the architecture, and the number of features in the feature model related to that service domain.

After developing the initial version of the PLAGEMIS feature model, the next step we made was analyzing existing products. We followed a systematic methodology that involved conducting a search of publicly accessible dashboards for mobility management and evaluating the functionality provided to determine whether it was included in the feature model. If the functionality was not already included in the feature model, we evaluated its relevance for PLAGEMIS to determine whether it should be considered for inclusion. To document our process, we created a table (published in an online appendix⁴) that lists each dashboard and its features, as well as the features that were not included in our feature model. We used the table to refine the feature model that we had developed based on the ISO 14813-1 standard.

During this step, we found several features (e.g., accessibility/walkability and speeding ticket functionality) that were not explicitly covered by ISO 14813-1, but we deemed them essential for PLAGEMIS. Therefore, we added these features to our feature model. Moreover, we discovered certain aspects of the ISO 14813 standard that we had initially considered essential for our SPL. However, upon further examination, we concluded that they were irrelevant. Examples of such aspects include demand management, corridor management, and alternative route planning. Accordingly, we decided to exclude these features from our feature model to avoid unnecessary complexity. Finally, we identified certain features listed in the ISO 14813-1 standard that we initially overlooked but later deemed relevant for PLAGEMIS (e.g., weight-in-motion, signal monitoring, and transportation cost) and included them in our feature model. We summarize the modifications, deletions, and additions we made to the feature model in the column *Dashboard Refined* of Table 2.

After refining the feature model based on the extracted dashboard features, we recognized the importance of obtaining feedback from domain experts to validate the model's completeness and accuracy. Thus, we organized a meeting with three researchers from our group to present the updated feature model and receive their feedback. The specialists thoroughly reviewed and scrutinized the

feature model to ensure that it includes all the critical features required for developing effective mobility solutions. The experts pointed out to us, among other things, that the dashboards we chose for the study consider mobility aspects for urban areas only, and that they believed that it would be essential to include features for mobility in rural environments. Some of these considerations were: information on scenic drives and country roads, information on observing wildlife, as well as information on hiking and camping sites. Their insightful feedback and suggestions helped us detect gaps and deficiencies in our feature model, and we improved it further based on their comments. Finally, we developed a final version of the feature model that includes all the necessary features for mobility systems, which was validated by the experts to confirm that it meets all the necessary requirements and guidelines for efficient and successful mobility systems. The validated feature model served as the basis for our new SPL.

4.2 Second Iteration: Review of the GIS SPL

In the second iteration of our action research methodology, we focused on analyzing the existing GIS SPL to determine whether it could be reused for the new project or whether it needed to be deprecated. After a thorough evaluation, we concluded that it was necessary to discontinue the existing SPL, due to its outdated frontend technology. The existing SPL uses Leaflet, which renders maps using SVG and HTML DOM, but this technology has been found to be inefficient compared to the newer and faster WebGL technology. Additionally, the frontend is implemented using Vue.js 2, whereas the most recent version of Vue.js offers enhanced performance and a more satisfying user experience for users. We also compared the PLAGEMIS SPL architecture to the existing GIS SPL architecture, and we found that the existing SPL is a subset of PLAGEMIS. This is because, although the scope of PLAGEMIS are larger MISs, it must include a web-based geographic information system, which is precisely the scope of the existing GIS SPL. So, we learned that the existing SPL is somewhat outdated and subsumed by the new SPL, leading to the decision that it would be better to design a new SPL architecture rather than trying to fit new features to the legacy SPL.

To prepare the retirement process, we decided to analyze the level of coherence between the annotated code and the feature model. We were looking for two scenarios in which the annotated code and feature model could be out of sync: features described in the SPL, but never implemented in the annotated code (i.e., *dangling features*); and features implemented in the code, but never added to the feature model (i.e., *orphaned code*). In addition, we also looked for features defined in the SPL that were never utilized in any SPL-generated product specification (i.e., *unused features*). In Table 3, we summarize the outcomes of this analysis.

Regarding *dangling features*, we conducted an extensive search of the codebase to determine whether each feature still existed in the code. We identified a total of 18 dangling features in our analysis. An example for such a feature is the Google Maps API support. Unfortunately, Google monetized the API before we could implement this feature, so it was never completely implemented. Another example of a dangling feature that we discovered is the support of multiple Database Management Systems (DBMSs) that was originally planned. However, as the SPL evolved, we found

⁴<https://zenodo.org/record/8107141>

Table 3: SPL feature model and products review.

Review Stage		Features
Total Features		117
Dangling features		18
Orphaned code		0
Unused features		14
Overlapping		4
Decision	Eliminate	4
	Persist	0

that Postgres/PostGIS was sufficient for the product requirements. Therefore, although the feature remains in the feature model, no annotated code references it. We also checked for *orphaned code*, but found no features in the annotated code that was not defined in the feature model. This was not surprising to us, since the evolution process that we applied prioritizes consistency between the added functionalities and the feature model. To identify *unused features*, we revised 31 product specifications and compared them to the SPL feature model to determine which features were never used in any product. This provided us with an indication regarding which features could be deprecated due to their lack of usage in previous years (e.g., the aforementioned Google Maps API, storing information in DBMSs other than Postgres/PostGIS, using a map server different than GeoServer, or functionality to geolocate documents and CSV files). Finally, we compared the features identified in the previous steps to identify overlaps, which were strong candidates for elimination due to their absence from the annotated code and product requirements.

Following this in-depth review and evaluation, we decided to discontinue the current SPL (i.e., learning after the second iteration). However, we also determined that it would be useful to integrate the feature model of the retired SPL into the feature model of PLAGEMIS. Additionally, we envisioned that at least some of the old backend components could be reused in the new SPL.

4.3 Iteration 3: Implementation of the new SPL

During the third iteration, we focused on implementing the new SPL by re-engineering some assets of the existing SPL with those of the new one. The implementation process began with a clear understanding of the architecture of the new SPL, the revised architecture of the existing one, and a list of products that needed to be migrated to the new SPL. We identified two challenges during the diagnosis step. First, merging the feature models and assets of the existing and new SPLs was cumbersome. Second, migrating the existing products to the new SPL posed problems.

Regarding the integration of the architectures, we conducted a thorough comparison of the feature models and the architectures. During this process, we kept a record of the operations we performed on the feature model of the existing SPL. In Table 4, we enumerate the operations that we required. For each operation, we provide a short description and an example of the transformation applied to the feature model. In Table 5, we describe, for each operation, the implications on the products, the SPL code, and the possibility of performing the operation automatically or

semi-automatically when the existing products are migrated. We used these tables as a reference during the implementation.

After completing the task of integrating the architectures, the next step was the migration of existing products. This task involved adapting the existing products to work within the new SPL using the operations defined previously. For each existing product, we performed the migration of the product configuration, generated a new version of the product with the new SPL, and verified that there were no errors in the product.


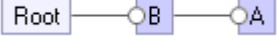


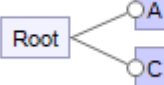
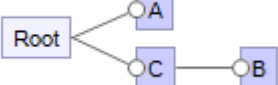


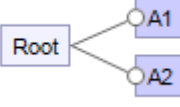



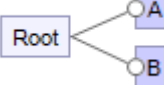
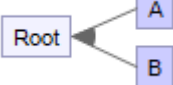


In Table 6, we provide insights into the steps we took during the implementation of the new SPL. The first row shows the number of features defined in the new SPL after the first iteration. The second row represents the total number of features in the existing SPL before any transformation. The third row shows the number of operations required to integrate the feature models of the existing and new SPLs. The fourth row displays the number of features present in the final feature model after we performed all transformations. We observed that not all of the defined operations were required, and we did not have to apply manual changes to the product specifications or the annotated code. This was mainly because any features that we removed from the previous SPL were not included in the product specifications or referenced in the annotated code.

4.4 Iteration 4: Definition of the Methodology

During our last iteration, we aimed to formalize the steps that we carried out previously to create a methodology that can be more easily replicated for future projects. We display the final methodology in Figure 2, outlining the steps for reusing and retiring the architecture of an SPL. Next, we describe each of the steps.

- (1) **Definition of the new SPL.** Our methodology begins with gathering the requirements for the new SPL. These requirements serve as input for defining the SPL and guiding the subsequent steps. This definition task consists of three steps that are performed in parallel and iteratively. The first step involves analyzing existing products, the second step is analyzing the reference architecture, and the third step involves consulting domain experts and stakeholders to gather knowledge about the domain. Iterations are performed to refine the SPL architecture based on the outputs of each step. The goal is to match the SPL requirements with the features from existing products, with components and functionality from the reference architecture, and with the knowledge of experts; as well as to identify any gaps or conflicts between the new SPL architecture and the requirements. The output of the task is the architecture of the new SPL and a prioritization of the features based on their importance to the product family. Not surprisingly, this step is highly similar to typical domain analyses in SPL scoping methods or steps in established process models [6, 35, 45, 57]. In more detail, the three steps in this task are:
 - (a) **Analyze Existing Products.** This step involves analyzing the existing products in the new SPL domain to identify commonalities and variabilities among them. To comprehend the functionality, performance, and design of each product, this also involves reviewing product specifications and other pertinent artifacts. The goal is to identify

Table 4: Operations we used for transforming the feature model.

Operation	Description	FM Before	FM After
Create(A,B)	Creates a new feature <i>A</i> as a child of feature <i>B</i> .		
Rename(A, A')	Changes the name of feature <i>A</i> to <i>A'</i> .		
Move(B, C)	Changes the parent of feature <i>B</i> to <i>C</i> .		
Split(A, [A1,A2])	Substitutes feature <i>A</i> by two features <i>A1</i> and <i>A2</i> .		
Fuse([A1,A2], A)	Combine the features <i>A1</i> and <i>A2</i> into a new feature <i>A</i> .		
Delete(A)	Removes feature <i>A</i> from the feature model.		
SetChildrenRelation(Root, or)	Sets the relationship of feature <i>Root</i> to its children to be of type <i>or</i> (the second argument could also be <i>xor</i> or <i>none</i>).		
SetProperty(A, mandatory, true)	Sets the <i>mandatory</i> property of <i>A</i> to be true. The second argument could also be <i>abstract</i> , and the third one could also be <i>false</i> .		

which traits and advantages are shared by all products and which are specific to each product. Using this information, we can define the scope of the new SPL.

- (b) **Analyze Reference Architectures.** This step requires analyzing the reference architectures that will be used as a basis for the new SPL to understand its key abstractions and concepts. To accomplish this, it is necessary to identify the design concepts and patterns used in the reference architectures by looking through the source code, architecture documentation, and other available data. Identifying which elements and resources from the reference design can be used in the new SPL and which ones need to be modified or replaced is the primary objective.
- (c) **Consider Experts Knowledge.** This step comprises eliciting the knowledge and expertise of domain experts, software architects, and other stakeholders who can provide valuable insights into the design and implementation of

the new SPL. Their input can help identify the most important features and assets to include in the new SPL, as well as potential challenges and risks.

- (2) **Feature and Asset Review of Existing SPL Architecture.** This task involves a two-fold task to review and analyze the existing SPL architecture. First, we must identify which features and assets of the existing SPL architecture are still relevant and can be reused in the new SPL, and which ones need to be retired because they are obsolete or not relevant. Second, we must analyze the coherence between the annotated code and the feature model. This involves identifying discrepancies between the annotated code and feature model, including dangling features, orphaned code, and unused features. The output of this task is a report detailing the reusable and obsolete features and assets in the existing SPL architecture, and a summary table of the discrepancies between annotated code and feature model that represent candidates for retirement.

Table 5: Impact of each feature-model transformation operation.

Operation	Product Configuration	Annotated Code	Automation Level
Create	No changes.	No changes.	Automatic.
Rename	Update to use the new feature name.	Updated to use the new feature name.	Automatic.
Move	The product configurations must only be changed in specific cases (e.g., when the parent-children relation of the new parent is XOR and there is already a child selected in the configuration).	No changes.	Semi-automatic. The change in the product configuration must be done manually.
Split	Update the existing product configurations to use the new features and remove the original feature.	Update the annotated code and change the feature for the new features.	Automatic.
Fuse	Update the existing product configurations to use the new feature and remove the original features.	Update the annotated code and change the original features to the fused feature.	Semi-automatic. The product configurations can be changed automatically. The annotated code may have to be reviewed because it may have inconsistencies (e.g., if two exclusive and incompatible features are fused together, the resulting code may not be correct).
Delete	Remove the feature from the existing product configurations.	Remove the code that is annotated with the feature.	Automatic.
SetChildrenRelation	Depending on the change, the product configuration must be changed manually (e.g., a relationship that changes to <i>xor</i> may require that some features are removed to leave only one child selected).	The annotated code does not change.	Semi-automatic. Depending on the change, the modifications to the product configuration have to be done manually.
SetProperty	If the <i>mandatory</i> property changes to <i>true</i> , and the feature is not already included in the feature model, it must be included. In all other cases, the product configuration does not change.	The annotated code only changes if the <i>abstract</i> property changes to <i>true</i> because any annotated code with that feature should be removed or moved outside of the annotation. In the other cases, the annotated code does not change.	Semi-automatic. The mentioned cases will probably require a manual review of the product configuration and the annotated code.

Table 6: Number of features through all transformations.

Transformation Phases		
Features in the new SPL		83
Features in the existing SPL		117
Transformation operations	Create	83
	Delete	4
	SetProperty	8
	SetChildrenRelation	12
PLAGEMIS SPL features		200
Product configurations migrated		30
Manual changes		0

(3) **New SPL Implementation.** The final task involves integrating the new architecture with the existing one, detecting overlaps and conflicts, reconciling them, and adapting the assets of the existing SPL architecture to fit into the new architecture. Additionally, the task includes adapting the product configurations for the products of the old SPL to fit into the new feature model of the new SPL architecture. This task requires identifying which configurations need to be updated and modified and validating the migrated products. Specifically, the two steps of this task are to:

- (a) **Integrate Architectures.** This step comprises integrating the new SPL architecture with the existing SPL architecture to create the final SPL architecture. At this point, we

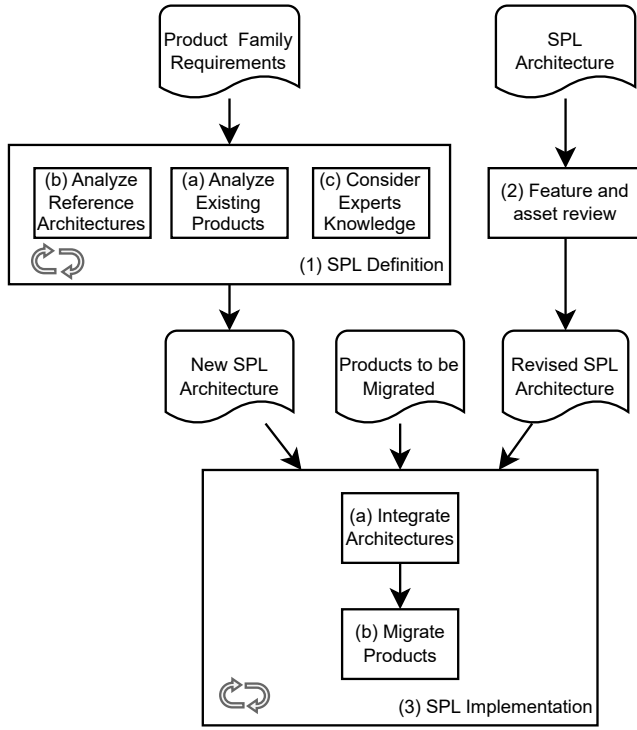


Figure 2: Our proposed methodology for retiring and replacing an SPL.

need to detect overlaps and conflicts between the architectures and reconciling them. The feature model should be reviewed to ensure that all the commonalities and variabilities of the products are captured and represented accurately. The assets of the existing SPL architecture should be refactored and adapted to fit into the new architecture, and any new assets required by the reference architecture should be developed.

- (b) **Migrate Products.** In this step, we migrate the products from the existing SPL to the new SPL by adapting their configurations to the new architecture. This entails determining which product configurations require updating and adjusting them to conform to the new feature model. The products should be verified after the migration to ensure they still meet their functional and non-functional requirements. Such a validation should be done for any relevant aspect of the products, including usability, performance, and functionality.

Overall, this process provides a guideline for organizations to retire and replace one SPL with a new one.

5 DISCUSSION AND LEARNING

In this section, we first answer and discuss our research questions. Furthermore, we summarize our current experiences with writing the GIS SPL and replacing it with the PLAGEMIS SPL.

5.1 Answering the Research Questions

To answer **RQ₁**, how to retire an SPL and replace it with a new one, we have developed a comprehensive methodology that outlines the critical steps and considerations required for a successful transition (cf. Figure 2). Our methodology focuses on analyzing the existing SPL, identifying its strengths and limitations, and determining the need for a new SPL based on the identified gaps. Additionally, we outline the process for selecting a new domain and architecture, as well as integrating the assets of the existing and new SPLs. Through our methodology, we provide a structured way for managing the retirement of an SPL and creating a new one, ensuring that the transition is efficient and effective. This methodology can guide other practitioners in this scenario, and it highlights how researchers can support them doing so.

To answer **RQ₂**, what are the challenges of retiring and replacing an SPL, we experienced several issues that must be addressed to ensure a smooth and successful migration. Migrating existing products to the new SPL, while ensuring they continue to function is arguably the biggest challenge. When dealing with numerous products, each with unique features and specifications, this can become especially complicated. Additionally, altering the feature model can have significant unintended consequences that necessitate modifying the annotated code—which is not always a simple task. It is particularly problematic to perform the migration process if it must be completed without affecting the functionality of the current products. In this paper, we have presented our methodology with a set of transformation operations and their implications on the product configurations and annotated code. These operators can help address the challenges of retiring and replacing an SPL, facilitating a successful transition.

To answer **RQ₃**, how to decide whether to reuse legacy SPL artifacts, we determined this through thorough analyses of various factors. The most important method to do so is to assess the technology used in the SPL and to determine whether it is still beneficial and pertinent. By collecting and examining relevant metrics (e.g., the frequency with which a feature is used in the products, the impact of removing a feature on product quality, the costs of maintaining a feature), we were able to determine the significance of each feature in the products developed and maintained by the company. While we provided some hints on how to make such decisions, we feel that more studies are needed to provide concrete guidance on what trade-offs between reusing a feature or implementing it anew practitioners must consider.

5.2 Lessons Learned

During our case study, we obtained particularly two further insights, which we detail in the following.

Annotation-Based Variability Mechanisms Facilitate Analyses. One benefit we experienced with our annotation-based implementations of the SPLs is that it is simpler to perform design operations and convert features between models. When the code is annotated and referred to in a feature model, maintaining a precise connection between the two is easier during transformations. We have to take into account that composition-based SPLs may profit differently, since they demand a more complicated but also a direct

mapping between a feature and its implementation. From our experience and similar findings in other studies [3, 16, 32], we argue that composition-based implementation techniques may make it more challenging to preserve a clear and consistent connection between code and feature models during the transformations and re-engineering we employed. As a result, it may be necessary in some circumstances to use a variety of strategies to achieve the desired outcomes.

Formalizing Transformations in the Process is Key. In our process for retiring an SPL and creating a new one, we have taken great care to formalize the transformations required to move from one feature model to another. By formalizing the transformations required to move from one feature model to another, we have established a clear and repeatable process for transitioning between SPLs. This is a significant benefit of our process, because it allows to undo all executed steps if necessary. During our case study, we experienced that it can be immensely helpful to return to a previous state, for instance, to fix errors more easily or revisit previous states for inspection. Additionally, while it is possible to specify additional operations, our set of operations for feature model transformation was sufficient for transforming our specific feature model.

6 CONCLUSION

In this paper, we outlined a process for retiring an existing SPL and replacing it with a new one. Through a case study, we identified and described the choices involved in this process and their effects. We decided to further explore this scenario using an action-research-like methodology, since there were no in-depth studies on this SPL evolution scenario in the literature. Via four iterations of our methodology, each centered on a distinct set of actions, we derived our process. In the first iteration, we examined requirements and specifications for the new SPL. In the second iteration, we determined whether the existing SPL could be reused. In the third iteration, we combined assets of both SPLs. In the fourth and final iteration, we defined our process for deprecating the existing SPL and developing a new one based on its assets, taking into account our previous experiences. The process we developed is aimed at providing a systematic way for deprecating and replacing an existing SPL, and for determining whether to reuse legacy SPL artifacts. For both practitioners and researchers in this field, our experiences offer perspectives and experiences from an industrial case study. We also describe the difficulties we experienced and how these may impact future studies and applications. Overall, we think that the results of our study are a helpful manual for software development teams dealing with similar circumstances.

Based on our current research, we think that additional analyses of the procedures and processes involved in integrating architectures and migrating products is necessary. It would be beneficial to conduct more case studies using our process as part of future work. So, we can improve our process and learn more about how it functions in various contexts by looking at more cases. In addition, we would be able to pinpoint problems that can impact everyone in the software engineering community as well as possible solutions. We also want to investigate the creation of automated tools for the migration and integration processes. Another significant future work would be to propose specific metrics for selecting which

SPL assets to retire. Although we provided some criteria for this decision-making process, it could benefit from more extensive and quantitative metrics. For instance, metrics that take into account the legacy assets' frequency of use, maintainability, and compatibility may be helpful in deciding whether to retire or reuse them. Further research into how retiring particular assets can impact the overall quality of the new (or an existing) SPL may reveal which assets are most essential to the success of a system.

ACKNOWLEDGMENTS

PID2021-122554OB-C33 (OASSIS): partially funded by MCIN/AEI/10.13039/501100011033 and EU/ERDF A way of making Europe; TED2021-129245B-C21 (PLAGEMIS): partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR; PDC2021-121239-C31 (FLATCITY-POC): partially funded by MCIN/AEI/10.13039/501100011033 and "NextGenerationEU"/PRTR; PID2020-114635R-B-I00 (EXTRACompact): partially funded by MCIN/AEI/10.13039/501100011033; MAGIST: PID2019-105221RB-C41, partially funded by MCIN/AEI/10.13039/501100011033; GRC: ED431C 2021/53, partially funded by GAIN/Xunta de Galicia; CITIC is funded by the Xunta de Galicia through the collaboration agreement between the Department of Culture, Education, Vocational Training and Universities and the Galician universities for the reinforcement of the research centers of the Galician University System (CIGUS).

REFERENCES

- [1] Faheem Ahmed and Luiz F. Capretz. 2010. A Business Maturity Model of Software Product Line Engineering. *Information Systems Frontiers* 13, 4 (2010), 543–560. <https://doi.org/10.1007/s10796-010-9230-8>
- [2] Faheem Ahmed, Luiz F. Capretz, and Shahbaz A. Sheikh. 2007. Institutionalization of Software Product Line: An Empirical Investigation of Key Organizational Factors. *Journal of Systems and Software* 80, 6 (2007), 836–849. <https://doi.org/10.1016/j.jss.2006.09.010>
- [3] Jonas Åkesson, Sebastian Nilsson, Jacob Krüger, and Thorsten Berger. 2019. Migrating the Android Apo-Games into an Annotation-Based Software Product Line. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 103–107. <https://doi.org/10.1145/3336294.3342362>
- [4] Suilen H. Alvarado, Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, and Ángeles S. Places. 2020. Developing Web-Based Geographic Information Systems with a DSL: Proposal and Case Study. *Journal of Web Engineering* 19, 2 (2020), 167–194.
- [5] Sofia Ananieva, Sandra Greiner, Jacob Krüger, Lukas Linsbauer, Sten Grüner, Timo Kehrer, Thomas Kühn, Christoph Seidl, and Ralf Reussner. 2022. Unified Operations for Variability in Space and Time. ACM, 7:1–10. <https://doi.org/10.1145/3510466.3510483>
- [6] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines*. Springer. <https://doi.org/10.1007/978-3-642-37521-7>
- [7] Maider Azanza, Leticia Montalvillo, and Oscar Díaz. 2021. 20 Years of Industrial Experience at SPLC: A Systematic Mapping Study. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 172–183. <https://doi.org/10.1145/3461001.3473059>
- [8] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc DeBaud. 1999. PuLSE: A Methodology to Develop Software Product Lines. In *Symposium on Software Reusability (SSR)*. ACM, 122–131. <https://doi.org/10.1145/303008.303063>
- [9] Jan Bosch. 2002. Maturity and Evolution in Software Product Lines: Approaches, Artefacts and Organization. In *International Software Product Line Conference (SPLC)*. Springer, 257–271. https://doi.org/10.1007/3-540-45652-x_16
- [10] Jan Bosch and Petra M. Bosch-Sijtsema. 2011. Introducing Agile Customer-Centered Development in a Legacy Software Product Line. *Software: Practice and Experience* 41, 8 (2011), 871–882. <https://doi.org/10.1002/spe.1063>
- [11] Reidar Conradi and Bernhard Westfechtel. 1998. Version Models for Software Configuration Management. *ACM Computing Surveys* 30, 2 (1998), 232–282. <https://doi.org/10.1145/280277.280280>
- [12] Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, Ángeles S. Places, and Jennifer Pérez. 2017. Web-Based Geographic Information Systems SPLE: Domain

- Analysis and Experience Report. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 190–194. <https://doi.org/10.1145/3106195.3106222>
- [13] Alejandro Cortiñas, Miguel R. Luaces, Oscar Pedreira, and Ángeles S. Places. 2018. Generation of Web-Based GIS Applications through the Reuse of Software Artefacts. In *International Symposium on Web and Wireless Geographical Information Systems (W2GIS)*. Springer, 11–14. https://doi.org/10.1007/978-3-319-90053-7_2
 - [14] Krzysztof Czarnecki, Paul Grünbacher, Rick Rabiser, Klaus Schmid, and Andrzej Wąsowski. 2012. Cool Features and Tough Decisions: A Comparison of Variability Modeling Approaches. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 173–182. <https://doi.org/10.1145/2110147.2110167>
 - [15] David de Castro, Alejandro Cortiñas, Miguel R. Luaces, Óscar Pedreira, and Ángeles S. Places. 2022. Improving the Customization of Software Product Lines through the Definition of Local Deatures. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 199–209. <https://doi.org/10.1145/3546932.3547006>
 - [16] Jamel Debbiche, Oskar Lignell, Jacob Krüger, and Thorsten Berger. 2019. Migrating Java-Based Apo-Games into a Composition-Based Software Product Line. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 98–102. <https://doi.org/10.1145/3336294.3342361>
 - [17] Deepak Dhungana, Paul Grünbacher, Rick Rabiser, and Thomas Neumayer. 2010. Structuring the Modeling Space and Supporting Evolution in Software Product Line Engineering. *Journal of Systems and Software* 83, 7 (2010), 1108–1122. <https://doi.org/10.1016/j.jss.2010.02.018>
 - [18] Ivan do Carmo Machado, John D. McGregor, Yguaratá C. Cavalcanti, and Eduardo S. De Almeida. 2014. On Strategies for Testing Software Product Lines: A Systematic Literature Review. *Information and Software Technology* 56, 10 (2014), 1183–1199. <https://doi.org/10.1016/j.infsof.2014.04.002>
 - [19] Sławomir Duszynski, Jens Knodel, and Martin Becker. 2011. Analyzing the Source Code of Multiple Software Variants for Reuse Potential. In *Working Conference on Reverse Engineering (WCRE)*. IEEE, 303–307. <https://doi.org/10.1109/wcre.2011.44>
 - [20] Ulrik Eklund and Håkan Gustavsson. 2013. Architecting Automotive Product Lines: Industrial Practice. *Science of Computer Programming* 78, 12 (2013), 2347–2359. <https://doi.org/10.1016/j.scico.2012.06.008>
 - [21] Emelie Engström and Per Runeson. 2011. Software Product Line Testing - A Systematic Mapping Study. *Information and Software Technology* 53, 1 (2011), 2–13. <https://doi.org/10.1016/j.infsof.2010.05.011>
 - [22] Stefan Ferber, Jürgen Haag, and Juha Savolainen. 2002. Feature Interaction and Dependencies: Modeling Features for Reengineering a Legacy Product Line. In *International Software Product Line Conference (SPLC)*. Springer, 235–256. https://doi.org/10.1007/3-540-45652-X_15
 - [23] Thomas S. Fogdal, Helene Scherrebeck, Juha Kuusela, Martin Becker, and Bo Zhang. 2016. Ten Years of Product Line Engineering at Danfoss: Lessons Learned and Way Ahead. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 252–261. <https://doi.org/10.1145/2934466.2934491>
 - [24] Cristina Gacek and Michalis Anastasopoulos. 2001. Implementing Product Line Variabilities. In *Symposium on Software Reusability (SSR)*. ACM, 109–117. <https://doi.org/10.1145/375212.375269>
 - [25] ISO 14813-1:2015(E). 2015. Intelligent transport systems – Reference model architecture(s) for the ITS sector – Part 1: ITS service domains, service groups and services.
 - [26] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Technical Report CMU/SEI-90-TR-21. Carnegie Mellon University.
 - [27] Mahvish Khurum and Tony Gorschek. 2009. A Systematic Review of Domain Analysis Solutions for Product Lines. *Journal of Systems and Software* 82, 12 (2009), 1982–2003. <https://doi.org/10.1016/j.jss.2009.06.048>
 - [28] Peter Knauber, Jesus Bermejo, Günter Böckle, Julio C. S. do Prado Leite, Frank J. van der Linden, Linda M. Northrop, Michael Stark, and David M. Weiss. 2002. Quantifying Product Line Benefits. In *International Workshop on Software Product-Family Engineering (PFE)*. Springer, 155–163. https://doi.org/10.1007/3-540-47833-7_15
 - [29] Heiko Koziol, Thomas Goldschmidt, Thijmen de Gooijer, Dominik Domis, Stephan Sehestedt, Thomas Gamer, and Markus Aleksy. 2016. Assessing Software Product Line Potential: An Exploratory Industrial Case Study. *Empirical Software Engineering* 21, 2 (2016), 411–448. <https://doi.org/10.1007/s10664-014-9358-0>
 - [30] Charles W. Krueger. 2002. Easing the Transition to Software Mass Customization. In *International Workshop on Software Product-Family Engineering (PFE)*. Springer, 282–293. https://doi.org/10.1007/3-540-47833-7_25
 - [31] Jacob Krüger. 2021. *Understanding the Re-Engineering of Variant-Rich Systems: An Empirical Work on Economics, Knowledge, Traceability, and Practices*. Ph.D. Dissertation. Otto-von-Guericke University Magdeburg. <https://doi.org/10.25673/39349>
 - [32] Jacob Krüger and Thorsten Berger. 2020. Activities and Costs of Re-Engineering Cloned Variants Into an Integrated Platform. In *International Working Conference on Variability Modelling of Software-Intensive Systems (VaMoS)*. ACM, 21:1–10. <https://doi.org/10.1145/3377024.3377044>
 - [33] Jacob Krüger and Thorsten Berger. 2020. An Empirical Analysis of the Costs of Clone- and Platform-Oriented Software Reuse. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 432–444. <https://doi.org/10.1145/3368089.3409684>
 - [34] Jacob Krüger, Wolfram Fenske, Thomas Thüm, Dirk Aporius, Gunter Saake, and Thomas Leich. 2018. Apo-Games - A Case Study for Reverse Engineering Variability from Cloned Java Variants. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 251–256. <https://doi.org/10.1145/3233027.3236403>
 - [35] Jacob Krüger, Wardah Mahmood, and Thorsten Berger. 2020. Promote-pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 2:1–12. <https://doi.org/10.1145/3382025.3414970>
 - [36] Elias Kuitert, Jacob Krüger, Sebastian Krieter, Thomas Leich, and Gunter Saake. 2018. Getting Rid of Clone-And-Own: Moving to a Software Product Line for Temperature Monitoring. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 189–189. <https://doi.org/10.1145/3233027.3233050>
 - [37] Miguel A. Laguna and Yania Crespo. 2013. A Systematic Mapping Study on Software Product Line Evolution: From Legacy System Reengineering to Product Line Refactoring. *Science of Computer Programming* 78, 8 (2013), 1010–1034. <https://doi.org/10.1016/j.scico.2012.05.003>
 - [38] Dong Li and Carl K. Chang. 2009. Initiating and Institutionalizing Software Product Line Engineering: From Bottom-Up Approach to Top-Down Practice. In *Annual Computer Software and Applications Conference (COMPSAC)*. IEEE, 53–60. <https://doi.org/10.1109/compsac.2009.17>
 - [39] Robert Lindorf, Jacob Krüger, Erik Herzog, and Thorsten Berger. 2021. Software Product-Line Evaluation in the Large. *Empirical Software Engineering* 26, 30 (2021), 1–41. <https://doi.org/10.1007/s10664-020-09913-9>
 - [40] Maira Marques, Jocelyn Simmonds, Pedro O. Rossel, and Maria Cecilia Bastarrica. 2019. Software Product Line Evolution: A Systematic Literature Review. *Information and Software Technology* 105 (2019), 190–208. <https://doi.org/10.1016/j.infsof.2018.08.014>
 - [41] Najam Nazar and T. M. J. Rakotomahefa. 2016. Analysis of a Small Company for Software Product Line Adoption — An Industrial Case Study. *International Journal of Computer Theory and Engineering* 8, 4 (2016), 313–322. <https://doi.org/10.7763/ijcte.2016.v8.1064>
 - [42] Damir Nešić, Jacob Krüger, Ștefan Stănculescu, and Thorsten Berger. 2019. Principles of Feature Modeling. In *Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 62–73. <https://doi.org/10.1145/3338906.3338974>
 - [43] Femi G. Olumofin and Vojislav B. Mišić. 2007. A Holistic Architecture Assessment Method for Software Product Lines. *Information and Software Technology* 49, 4 (2007), 309–323. <https://doi.org/10.1016/j.infsof.2006.05.003>
 - [44] Juliana Alves Pereira, Sebastian Krieter, Jens Meinicke, Reimar Schröter, Gunter Saake, and Thomas Leich. 2016. FeatureIDE: Scalable Product Configuration of Variable Systems. In *International Conference on Software Reuse (ICSR)*. Springer, 397–401. https://doi.org/10.1007/978-3-319-35122-3_27
 - [45] Klaus Pohl, Günter Böckle, and Frank J. van der Linden. 2005. *Software Product Line Engineering*. Springer. <https://doi.org/10.1007/3-540-28901-1>
 - [46] Rick Rabiser, Klaus Schmid, Martin Becker, Goetz Botterweck, Matthias Galster, Iris Groher, and Danny Weyns. 2018. A Study and Comparison of Industrial vs. Academic Software Product Line Research Published at SPLC. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 14–24. <https://doi.org/10.1145/3233027.3233028>
 - [47] Luisa Rincón, Raúl Mazo, and Camille Salinesi. 2019. Analyzing the Convenience of Adopting a Product Line Engineering Approach: An Industrial Qualitative Evaluation. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 90–97. <https://doi.org/10.1145/3307630.3342418>
 - [48] Julia Rubin, Krzysztof Czarnecki, and Marsha Chechik. 2015. Cloned Product Variants: From Ad-Hoc to Managed Software Product Lines. *International Journal on Software Tools for Technology Transfer* (2015), 627–646. <https://doi.org/10.1007/s10009-014-0347-9>
 - [49] Klaus Schmid, Isabel John, Ronny Kolb, and Gerald Meier. 2005. Introducing the PuLSE Approach to an Embedded System Population at Testo AG. In *International Conference on Software Engineering (ICSE)*. ACM, 544–552. <https://doi.org/10.1145/1062455.1062552>
 - [50] Klaus Schmid and Martin Verlage. 2002. The Economic Impact of Product Line Adoption and Evolution. *IEEE Software* 19, 4 (2002), 50–57. <https://doi.org/10.1109/ms.2002.1020287>
 - [51] Miroslaw Staron. 2020. *Action Research in Software Engineering*. Springer. <https://doi.org/10.1007/978-3-030-32610-4>
 - [52] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the Truth: Benchmarking the Techniques for the Evolution of Variant-Rich Systems. In *International Systems and Software Product Line Conference (SPLC)*. ACM, 177–188. <https://doi.org/10.1145/3336294.3336302>
 - [53] Mikael Svahnberg and Michael Mattsson. 2002. Conditions and Restrictions for Product Line Generation Migration. In *International Workshop on Software*

- Product-Family Engineering (PFE)*. Springer, 143–154.
- [54] Thomas Thüm, Sebastian Krieter, and Ina Schaefer. 2018. Product Configuration in the Wild: Strategies for Conflicting Decisions in Web Configurators. In *International Configuration Workshop (ConfWS)*. CEUR-WS.org, 1–8.
 - [55] Frank J. van der Linden. 2002. Software Product Families in Europe: The Esaps & Café Projects. *IEEE Software* 19, 4 (2002), 41–49. <https://doi.org/10.1109/ms.2002.1020286>
 - [56] Frank J. van der Linden. 2005. *Family Evaluation Framework: Overview & Introduction*. Technical Report PH-0503-01. Philips.
 - [57] Frank J. van der Linden, Klaus Schmid, and Eelco Rommes. 2007. *Software Product Lines in Action*. Springer. <https://doi.org/10.1007/978-3-540-71437-8>
 - [58] Claes Wohlin and Per Runeson. 2021. Guiding the Selection of Research Methodology in Industry–Academia Collaboration in Software Engineering. *Information and Software Technology* 140 (2021), 106678:1–14. <https://doi.org/10.1016/j.infsof.2021.106678>
 - [59] Kentaro Yoshimura, Dharmalingam Ganesan, and Dirk Muthig. 2006. Assessing Merge Potential of Existing Engine Control Systems into a Product Line. In *International Workshop on Software Engineering for Automotive Systems (SEAS)*. ACM, 61–67. <https://doi.org/10.1145/1138474.1138485>
 - [60] Bo Zhang, Slawomir Duszynski, and Martin Becker. 2016. Variability Mechanisms and Lessons Learned in Practice. In *International Workshop on Conducting Empirical Studies in Industry (CESI)*. ACM, 14–20. <https://doi.org/10.1145/2897045.2897048>