SPLReePlan - Automated Support for Software Product Line Reengineering Planning

Luciano Marchezan
Johannes Kepler University Linz
Linz, Austria

Wesley K. G. Assunção
Pontifical Catholic University of Rio de Janeiro
Rio de Janeiro, RJ, Brazil

João Carbonell
Federal University of Pampa
Alegrete, RS, Brazil

Elder Rodrigues
Federal University of Pampa
Alegrete, RS, Brazil

Maicon Bernardino
Federal University of Pampa
Alegrete, RS, Brazil

Fábio Basso
Federal University of Pampa
Alegrete, RS, Brazil

ABSTRACT

The extractive adoption of Software Product Lines (SPL) relies on the reuse of the already developed systems, employing a reengineering process. However, due to the diversity of options found in the daily practice of SPL development, rigorous planning of scenarios is critical to perform SPL reengineering. This diversity is the result of different organizational aspects, such as team experience and product portfolio. Hence, a proper planning process must consider technical and organizational aspects, however, most existing studies in the field do not take into account organizational aspects of the companies. In this work, we present SPLReePlan, an automated framework to aid the SPL reengineering planning taking into account technical and organizational aspects. Our framework is supported by a web-based tool, ready to be used in the industry. To investigate how flexible is SPLReePlan to support the SPL reengineering planning in diverse situations, we extracted eight different scenarios from the SPL literature, which are used as input for the evaluation of SPLReePlan. The results indicate that SPLReePlan can be satisfactorily customized to a variety of scenarios with different artifacts, feature retrieval techniques, and reengineering activities. As a contribution, we discuss the lessons learned within the evaluation, and present challenges that were faced, being a source of information for tool builders or motivating new studies.

CCS CONCEPTS

• **Software and its engineering → Software product lines**.

KEYWORDS

software product lines, reengineering process, variability management, automated support

ACM Reference Format:
Luciano Marchezan, Wesley K. G. Assunção, João Carbonell, Elder Rodrigues, Maicon Bernardino, and Fábio Basso. 2021. SPLReePlan - Automated Support for Software Product Line Reengineering Planning. In *15th Brazilian Symposium on Software Components, Architectures, and Reuse (SBCARS '21), September 27-October 1, 2021, Joinville, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3483899.3483902

SBCARS '21, September 27-October 1, 2021, Joinville, Brazil

ACM ISBN 978-1-4503-8419-3/21/09…$15.00
https://doi.org/10.1145/3483899.3483902

1 INTRODUCTION

Software Product Line Engineering (SPLE) is an industry-proven software development approach that relies on software reuse to enable companies to more effectively and efficiently develop their product portfolios [37]. The most common way to adopt SPLE is by employing an extractive strategy [21]. Companies usually have many system variants developed using opportunistic reuse, which are the basis for the SPLE activities [5]. Extractive adoption of Software Product Lines (SPL) is conducted by a reengineering process that encompasses the identification and extraction of common and variable features [23]. This process is divided into three phases, namely detection, analysis, and transformation [5]. A proper planning[1] of all these steps is paramount for the success of the SPL adoption. Furthermore, SPL evolved as any traditional software, then the potential evolution of the re-engineered SPL also should be taken into consideration as part of the reengineering planning.

Despite the great number of studies on the topic of SPL reengineering [5, 23], there are still limitations. Firstly, *the studies in the literature mostly focus on technical aspects of the variants*, namely identification of commonalities and variabilities among variants implementation [13, 31], without taking into account organizational aspects, *e.g.*, team experience. Considering only technical aspects hampers the application of the SPL reengineering process in real-world scenarios. Also, considering only technical aspects limits the decision-making concerning feature selection and product development, as decisions are usually impacted by organizational aspects, which vary according to the scenario. Secondly, *the vast majority of approaches are inflexible and strictly designed for a specific scenario*, hampering their use in other companies and systems. The most promising approach that we found is called PAxSPL [27]. PAxSPL is a process supported by guidelines that help users to customize a feature retrieval process for their context. Nevertheless, this process still has the limitation described in the following. Thirdly, *existing automated support is mainly based on academic prototypes developed for validating an approach, not mature enough to be used in practice* [23]. These tool prototypes, although important to be used as

[1] In this work, SPL reengineering planning refers to the activities for initiating, instantiating, documenting, and analyzing the activities, techniques, methods, and tools to be used along all the reengineering process.

a proof-of-concept, are not designed with a focus on the end-user, limiting its application in real scenarios.

Based on the aforementioned limitations, in this work, we present ***SPL Ree**ngineering **Plan**ning (SPLReePlan)*, a framework to support the initiation, instantiation, documentation, and analysis of the SPL reengineering. SPLReePlan. The framework supports the SPL reengineering planning process by implementing a web-based supporting tool developed considering end-user requirements. More specifically, we present a tool implemented based on i) SPL reengineering planning activities [27]; ii) limitations of similar studies found in the literature [5, 23], and iii) end-user requirements of similar tools identified in the literature [32]. SPLReePlan was evaluated with eight distinct scenarios extracted from the literature [28]. We instantiated these scenarios to better understand the benefits and drawbacks of SPLReePlan. The evaluation results indicate that our framework supports the planning of a variety of SPL reengineering scenarios, considering different artifacts (more than 20 types), techniques (at least five), and activities. The results also evidence that our framework organizes and generates a repository of documents that might be important for the SPL reengineering analysis and evolution. Still, we identified eight challenges, that were used to discuss the lessons learned when conducting the evaluation.

The main contributions of our work are: i) the SPLReePlan framework automated by a web-based supporting tool; ii) empirical results demonstrating how SPLReePlan can give support to the SPL reengineering planning in different scenarios; iii) eight challenges faced when customizing different SPL reengineering processes and the lessons learned with them, and iv) a repository of artifacts generated during the evaluation [25].

2 BACKGROUND AND MOTIVATION

In this section, we describe the SPL reengineering process, the planning of this process, and the motivation of our work.

Software Reengineering Process. The extractive adoption of SPLs is employed by a reengineering process [5, 23] that is composed of three phases [5]: i) *detection*, when variabilities and commonalities among variants, represented in the form of features, are identified and extracted throughout the use of feature retrieval techniques; ii) *analysis*, the information extracted is used to design and organize the features in a variability model, usually a Feature Model (FM); iii) *transformation*, variant artifacts linked to the features, such as source code and requirements, are refactored/modified to include variability mechanisms and create the SPL.

SPL Reengineering Planning. The SPL reengineering planning[1] consists of activities to deal with specific aspects of the organizational scenario in which the reengineering process will take place. Our work relies on the Prepare, Assemble and Execute (PAxSPL) process [27]. PAxSPL was designed to support the planning of feature retrieval during the detection and analysis phase of the reengineering process. This support is done by aiding the decisions related to the selection of strategies and techniques for feature retrieval. PAxSPL is composed of three phases. During the *Prepare* phase, information related to artifacts and documentation is collected. Then, during the *Assemble* phase, the information collected previously is analyzed to help the selection of techniques for the feature retrieval. These chosen techniques are instantiated into a generic process. The generic process consists of the main activities performed during the feature retrieval, observed in the literature [5]: *Extract*, *Categorize* and *Group*, activities performed with the features before the creation of an FM. Lastly, during the *Execute* phase, the feature retrieval is performed by executing the instantiated process and the feature artifacts are collected.

The Reengineering Process in Organizational Scenarios. The extractive adoption of SPLs in different organizational scenarios[2] is challenging [7]. In this sense, when performing the reengineering of legacy systems, companies may face different challenges understanding and dealing with features being identified, modeled and extracted [19]. We can find a variety of scenarios in the ESPLA catalog [28]. One of these scenarios is presented by Eyal-Salman *et al.* [13], in which object-oriented (OO) source code is used for extracting features by applying three different retrieval techniques, namely formal concept analysis (FCA) [16], latent semantic indexing (LSI) [12], and clustering. Another scenario is presented by Acher *et al.* [1] that uses dependency analysis, structural similarity, and clustering as techniques for retrieving FMs from plugin dependencies. Although both strategies were useful for their respective scenarios, their proposals can mostly be used in other scenarios that share the same characteristics. To address the aforementioned issue is important to plan the SPL reengineering considering all possible scenario variables. Most approaches, however, are designed to a specific scenario, and those that are extensible and customizable mostly consider only technical aspects [5]. However, organizational aspects, such as team experience and product portfolios, also must be taken into account during the reengineering planning. Furthermore, the majority of existing automated support for the reengineering process are prototypes tools used as a proof of concept. Hence, these tools were not developed to be applied by end-users. This impacts the flexibility and scalability of the tool.

3 SPLREEPLAN

The SPL Reengineering Planning framework (SPLReePlan) proposed in this work aims at overcoming the limitations observed in the literature. Our framework provides automated support to SPL reengineering planning activities with the use of a web-based tool.[3] The SPLReePlan requirements were defined based on three sources of information: i) the SPL reengineering planning activities [27]; ii) limitations evidenced in the literature [5, 23, 27], and iii) end-user requirements of similar tools [24, 32].

3.1 Framework Structure

Figure 1 presents SPLReePlan that is composed of four steps:

1: Initiation: in this step, software engineers collect technical and organizational information to be used for defining a customized process and support decision-making during the SPL reengineering planning. This information, depending on the availability of artifacts, may include team experience, domain information, requirements, design decisions, design models, development artifacts, and technological information.

[2] In this work, an organizational scenario is "a certain group of engineers, having a set of system variants, knowing a specific set of techniques for analyzing features from a determined set of artifacts".

[3] Tool is available for use at our website http://splreeplan.herokuapp.com

2: Instantiation: based on information collected in the previous step, a customized process is defined, generating decisions related to the feature retrieval process. For creating this retrieval process, our framework recommends feature retrieval techniques based on heuristics computed using the information previously collected. These heuristics are formulated based on PAxSPL's guidelines on retrieval techniques, where each technique has a definition, priority order to be performed, type of input artifacts, possible outputs generated, examples of use, available tools, related techniques, and recommended scenarios. A technique will be higher recommended if: i) members' experience registered indicates whether the team has the knowledge to apply the technique; ii) types of artifacts registered can be used by the technique, *e.g.*, source code can be used by the clustering technique. Then, these techniques can be assembled, instantiating it into the feature retrieval process. Although the framework recommends the techniques, it is the user who decides which ones to use, and in which activities they should be used. Our framework also supports the management of the feature retrieval process, *i.e.*, managing the status of activities, roles, and documenting artifacts.

3: Documentation: this step of the framework supports the creation and management of the reengineering artifacts, composing an artifacts repository. This is done during the execution of the feature retrieval process. The users can change the status of an activity from *to do* to *doing*, which allows them to register artifacts generated in that activity. These artifacts can be stored in an online repository, *e.g.* GitHub, and in our tool the user registers the artifact name, type, file format, description, and the hyperlink for accessing it in its online repository. The user who registered the artifact in the tool will be assigned as its owner. After all artifacts from an activity have been registered, the user can change that activity's status to *done*. As our tool supports a multi-user environment, all other users involved in that project can check the current status of all activities, as well as collaborate to its execution by creating/editing artifacts.

4: Analysis: the artifacts repository can be used for reasoning about potential SPL evolution, as well as the generation of reports related to technical and organizational aspects. Since the repository contains artifacts generated during all phases of the framework, users can analyze their reengineering process as well as the results of the feature retrieval. The tool also supports registering an overall report about the project, with problems encountered and how they were resolved. This report can be used by companies to plan the improvement of the reengineering process for future projects. Users can also, for instance, use the SPLReePlan tool to generate configurations of possible products from the SPL[4], analyzing which features and artifacts are related to each product.

3.2 Supporting Tool

We developed our tool following an iterative development lifecycle. For that, we performed three iterations, and for each, we implement the requirements[5] of one source of information used to define our framework [5, 23, 24, 27, 32]. These iterations and their requirements are described in the following.

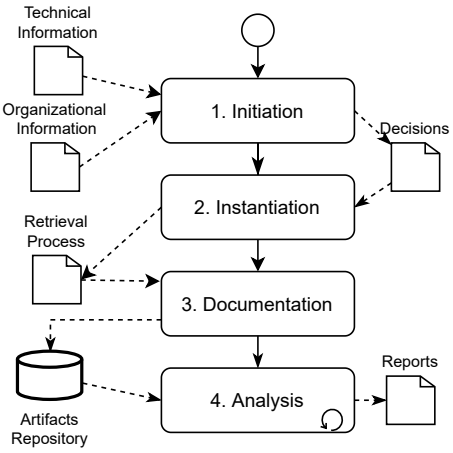


Figure 1: The SPLReePlan Framework

Requirements of the SPL reengineering planning: during the first iteration, we defined eight user stories (US) based on the requirements for the SPL reengineering planning activities. These activities are related to PAxSPL [27] process. The tool aims at aiding the SPL reengineering by structuring the way information is registered/managed, as well as by allowing multiple users to work in parallel in different activities.

It is important to clarify that the tool does not intend to automatically extract information from documents. The tool allows registering information, both technical and organizational. For instance, when collecting information about the team, each user can fill out a form in the tool related to his/her experience, skills, and knowledge in feature retrieval techniques. This information is used later when deciding which techniques will be applied. Furthermore, these decisions will also be registered in the tool, which leads to the analysis of the planning process itself in the future, aiming at understanding or evolving the company's organizational aspects.

Requirements based on limitations identified: for the second iteration, we defined the requirements considering limitations presented in a case study [27] and limitations of other tools discussed in secondary studies [5, 23]. The limitations reported in the case study were the basis for defining requirements. For instance, the need for organizing and versioning the artifacts generated during the SPL reengineering, *e.g.*, reports. In the case study, participants spent a considerable amount of time organizing artifacts [27]. This limitation can be mitigated with automated support to aid the management and execution of the process. This observation corroborates with findings of previous studies, that pointed out the lack of tools to support SPLE [5, 32]. Based on this analysis we concluded that the limitations were related to the difficulties that the users had to maintain a repository of artifacts well-structured and updated. Thus, we defined five USs to address these limitations. An example of functionality developed based on these requirements, is that when the feature retrieval is being performed, the tool allows

[4]This generation does not include source code as our framework focus on the analysis rather than the implementation of SPL.

[5]Our tool is open source and its documentation, including user stories, is available at our repository at https://tinyurl.com/splreeplan.

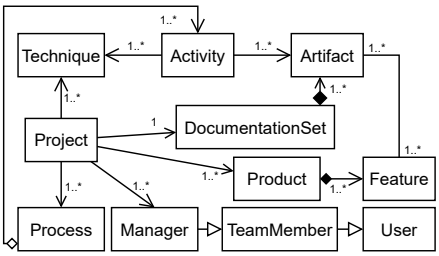


Figure 2: SPLReePlan Tool Class Diagram

the consistency verification (manually performed by another user) of the artifacts generated, *e.g.* a concept lattice generated by FCA.

Requirements of end-user: we also focused on increasing the reliability and applicability of our tool by defining end-user requirements accordingly to similar SPL tools. For this reason, during the third iteration, we sought in the literature to find empirical studies reporting important requirements for such tools. We found studies [24, 32] describing 17 functionalities used to compare SPL management tools.

Then, we analyzed these 17 functionalities to decide which would be relevant for SPLReePlan, as some of them were not suited to our reengineering planning tool, *e.g.* the source code generation. After this analysis, we defined seven USs based on the functionalities presented in [32]. By defining the US based on end-user requirements we can argue that our SPLReePlan tool is not only a prototype or proof-of-concept but a ready-to-use tool for application in real scenarios. An example of a functionality of our tool is to register the features retrieved from the system variants, their descriptions, relations, and types. Also, the artifacts from the legacy system can be used to generate a traceability matrix, tracing the origin from where the features were identified. Another functionality of our tool is to allow users to create FMs and configure products from it. After configuring a valid product, the configuration can be downloaded as an XML file, which is compatible with FeatureIDE.[6]

Architecture: Figure 2 illustrates the class diagram designed after considering all requirements mentioned. We defined a *Project* to be the central part of the tool. This *Project* is managed by one or more *Managers*, and it may have several *Processes* (*i.e.*, feature retrieval process) and one *Documentation Set*. This *Project* also is associated to several *Techniques*. The *Manager* is a specialization of a *Team Member*, which is a type of *User*. The *Documentation Set* is composed of *Artifacts*. All these classes cover the first phase of our framework, *Initiation*. Continuing, the *Process* is an aggregation of *Activities*. These activities may perform one or many *Techniques*. Then, we have the *Features* that are related to several *Artifacts*. These classes cover SPLReePlan *Instantiation* and *Documentation* steps. The *Product* class is a composition of *Features* and is related to a *Project*. Thus, several products can be generated from the features identified. These classes cover the last phase of SPLReePlan, *Analysis*.

[6]A well-known tool for developing SPL: https://featureide.github.io/

4 EVALUATION SETUP

We performed a study to evaluate SPLReePlan. In this section, we describe the goal, research questions (RQs), the dataset used, and the execution of this study.

4.1 Goal and Research Questions

The main goal of our study is to *evaluate how SPLReePlan supports and automates the SPL reengineering planning for different scenarios*. Thus, we aim to apply SPLReePlan to customize different reengineering processes in scenarios observed in related works. The RQs that guided the evaluation are:

RQ1. How does SPLReePlan automate the planning of activities, artifacts, and techniques, for the SPL reengineering? This RQ aims to evaluate how our framework can be instantiated to be applied to specific scenarios, automating the SPL reengineering planning process. We also analyze how the technical and organizational aspects are handled.

RQ2. How does SPLReePlan allow the customization of the reengineering planning in different scenarios found in the literature? This RQ aims to investigate how the customization across different scenarios can be addressed by SPLReePlan. Also, to identify issues and limitations when customizing SPLReePlan for varying scenarios, and how to handle them.

Our **RQ1** focuses on investigating how SPLReePlan adheres to individual scenarios, in terms of planing the initiation, instantiation, documentation, and analysis of the SPL reengineering process for specific cases. Thus, we want to identify what possible benefits emerged from using our framework to perform the aforementioned steps. For **RQ2** we aim to investigate how SPLReePlan adapts to a set of eight different scenarios, allowing a better understanding of the applicability and flexibility of our framework in multiple scenarios, as well as identifying challenges and limitations.

4.2 Dataset and Execution

The dataset used as input for this evaluation is composed of studies that report on the execution of an SPL reengineering process. We selected studies mapped in the ESPLA catalog [28], which is a collaborative catalog of case studies on SPL reengineering. We randomly selected the studies from the catalog, applying three criteria in each of them, for deciding to include it or not. The reason for defining these criteria is to select only studies that would represent a set of varying scenarios, which is required to answer RQ2. The study should be approved in all the following three criteria to be selected for this evaluation: i) *The study applies at least one retrieval technique supported by our framework*: we only considered studies that applied at least one retrieval technique among those covered by SPLReePlan. For example, in a study that applies three techniques, if one of them is covered by SPLReePlan, this criteria is satisfied, even that the other two techniques are not supported. ii) *The study presents a different scenario from other studies previously selected*: we want to guarantee that all selected studies have different scenarios from each other. We considered different scenarios when the study used at least one different retrieval technique, used at least one different input artifact, or had a different workflow when applying the feature retrieval techniques. iii) *The study evaluation protocol,*

dataset, and results are available online: Thus, we could extract their activities, artifacts, and techniques to conduct our evaluation.

After applying these criteria to 23 studies randomly selected from the ESPLA catalog, we composed a final set of eight studies, to be used as input for SPLReePlan evaluation.[7] Table 1 presents the data extracted from these studies. There is a variety of different artifacts from the studies, ranging from domain to development artifacts. Also, different techniques were used in these scenarios. There is also a different combination of techniques, which defines a different scenario.

We executed SPLReePlan in each scenario, *i.e.*, the artifacts, retrieval techniques, and activities, from the selected studies. Hence, once we started SPLReePlan execution for one scenario, we intend to observe if we can fully represent that scenario. We focus on evaluating how SPLReePlan automates the planning of the SPL reengineering process, comparing to the original process observed in the original study. For each study selected, we performed the following steps: i) identify and register inputs and output artifacts; ii) identify and register feature retrieval techniques used; iii) identify the feature retrieval activities and their workflow; and iv) execute SPLReePlan using the artifacts, techniques, and activities identified.

After executing these steps, we collect information to answer the RQs by analyzing: (1) The number of artifacts from the original study that were used by SPLReePlan (**RQ1 and RQ2**); (2) The retrieval techniques that were instantiated into SPLReePlan generic process (**RQ1 and RQ2**); (3) The activities that were instantiated into SPLReePlan generic process (**RQ1 and RQ2**); (4) The number of scenarios for which SPLReePlan was able to adapt to (**RQ2**); (5) The challenges and limitations found when adapting the scenarios using SPLReePlan (**RQ2**).

5 RESULTS AND ANALYSIS

In the following, we present the results and some artifacts generated by our tool during this execution. Due to space limitation, we only present four BPMN diagrams, and two report excerpts[7].

5.1 Scenario's Results

The first scenario was extracted from [13]. Figure 3(a) presents the BPMN[8] representation of the instantiated retrieval process. The five activities and their workflow were based on the original study, however, minor modifications were made due to some challenges faced (see Section 5.3). As the first scenario of the evaluation, this was unique in all aspects as no artifacts, techniques, and activities were previously executed. SPLReePlan gave support to the documentation of the different types of artifacts and activities. All the three techniques from this scenario (LSI, FCA, and clustering) were present in our tool, and SPLReePlan also gave support when selecting them. Figure 4(a) presents part of the report detailing one activity instantiated into the process as well as one input and one output artifact. In this case, the *Divide features with LSI* activity is detailed. This activity contains two input artifacts, the *Object-oriented* source code and the *Features descriptions*; and one output artifact, the *common and variable partitions*. This report is important as it demonstrates how the activities were performed during the reengineering and the artifacts generated from them. This can be used as the requirements documentation of the SPL, also allowing evolution planning. The second scenario was defined by extracting the information from [1]. Figure 3(b) presents the BPMN representation of the instantiated process. In this case, the process was composed of four activities which were described in Table 1. Figure 4(b) presents part of the process report, showing the first activity and artifacts used as input and output for it. In this case, a raw architecture from the systems should be extracted to generate an architectural model, which would then be refined into an FM. In comparison to the first scenario, this scenario uses different artifacts types, and two different techniques. The third technique, clustering, was also present in the first scenario, however, its application (an activity where it was applied) was different from scenario one.

The third scenario used in the evaluation was extracted from [2]. Figure 3(c) presents the instantiated process for the scenario, composed of three activities: extract building elements from object-oriented (OO) source code, extract commonalities and variabilities with FCA, and identify features using FCA and LSI. Considering the challenges faced when assembling this scenario, we faced two that were present in the previous scenarios. This scenario also has some similarities with those described earlier, as two retrieval techniques (one from the first and one from the second scenario) were also present here. We extracted the fourth scenario from [36]. In this scenario, the instantiated process was composed of four activities presented in Figure 3(d). These four activities are: extract component-based architecture, identify component variants, analyze commonality and variability, and identify architecture variability. The FCA and dependency analysis techniques were present in this scenario either. However, the ROMANTIC [9] approach also was used in the original study, however, such a technique was not supported by SPLReePlan as the framework supports the same techniques present in PAxSPL [27].

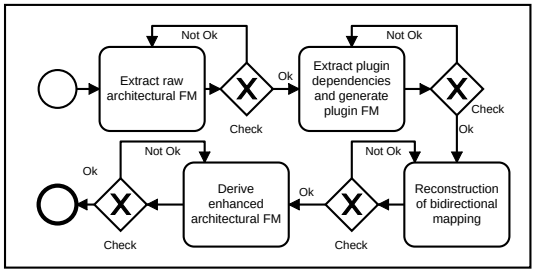
The fifth scenario used during the evaluation was extracted from [4], and its instantiated process was composed of three activities, perform a requirements similarity determination, cluster requirements to generate configuration, and merge configurations for all requirements. Considering the similarities of the scenario, the clustering, and LSI techniques were also present here. Also, the Vector-Space Model [34] technique was used, being the only scenario where it was present. The sixth scenario was extracted from [10], and its instantiated process was composed of four activities. These activities are: requirements elicitation, relationship graph construction, requirements clustering, and merging variability modeling. In this scenario, only the clustering technique was used, making it the most common retrieval technique.

The seventh scenario was defined by extracting the information from [31]. This scenario was composed of six activities, which are compile java source code, extract features dependencies from source code, construct feature distance matrix, cluster features based on dependency, convert feature tree into FM, and generate the description of FM in Feature Description Language (FDL) or PROLOG. The clustering and dependency analysis techniques were present once again. This scenario also shares characteristics concerning the artifacts used, which were object-oriented source code, the same as the first, third, and fourth scenarios. The eighth scenario was extracted

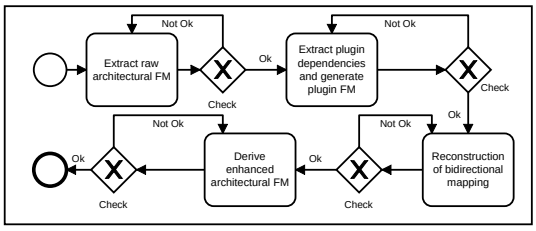
[7] Selected studies/scenarios, and evaluation artifacts are available in [25].
[8] All BPMN diagrams and reports were generated by our tool.

Table 1: Data Extracted from the Original Studies

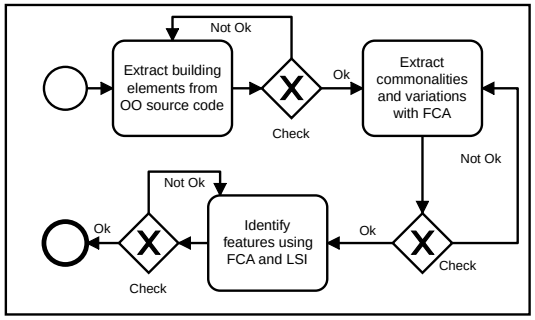
Ref	Artifacts	Techniques	Activities
[13]	object-oriented source code; feature descriptions	LSI; FCA; clustering;	i) Use LSI to divide features and classes into common and variable partitions; ii) fragment variable partitions into minimal disjoint sets using FCA; iii) derive code-topics from common class partition; iv) perform traceability links between features and their code-topics; v) determine which classes implement each feature.
[1]	150% architecture of the system; specification of the system plugins; system plugins dependencies; architectural FM; plugin FM; constraints mapping; enhanced architectural FM;	dependency analysis; structural similarity; clustering;	i) extraction of a raw architectural FM; ii) extraction of plugin dependencies to derive inter-feature constraints from inter-plugin constraints (plugin FM); iii) automatically reconstruction of the bidirectional mapping between the architect FM and plugin FM; iv) explore the mapping to derive enhanced architectural FM.
[2]	object-oriented source code; object-oriented building elements; commonalities and variations; blocks of variations; atomic blocks of variation;	LSI; FCA;	i) analyze OO source code to extract OO building elements; ii) commonalities and variations are extracted using FCA (blocks of variations); iii) blocks of variations are divided into atomic blocks and features are identified based on textual similarity using FCA and LSI.
[36]	object-oriented source code; component architecture; sets of component variants; concept lattice; architecture variability;	dependency analysis; FCA; ROMANTIC approach;	i) extract component-based architecture; ii) identity component variants; iii) use FCA to analyze the commonality and variability; iv) identify architecture variability.
[4]	requirement documents; requirements clusters; configurations; feature model;	LSI; Vector Space Model; clustering;	i) perform a requirements similarity determination; ii) abstract requirements clusters into a configuration; iii) merge configurations for all requirements.
[10]	individual requirements; requirements relationship graph; application feature trees; domain feature tree;	clustering;	i) requirements elicitation; ii) requirements relationship graph construction; iii) requirements clustering and hierarchical structure construction; iv) merging and variability modeling.
[31]	java source code; java .class files; dependency graph; feature distance matrix; cluster dendrogram; feature model; Feature model defined in FDL/Prolog;	dependency analysis; clustering;	i) compile Java source code using a standard Java compiler ; ii) extract feature dependencies from Java class files; iii) construct a feature distance matrix; iv) cluster features based on their dependency in a feature tree; v) convert a feature tree into a FM; vi) generate description of FM in FDL/Prolog.
[8]	architecture description; design documents; source code; user documentation; requirements specification; architecture requirements; common core assets; variable assets; SPL architecture;	dependency analysis;	i) identify requirements on the software architecture; ii) identify commonalities and variabilities; iii) restructure architecture; iv) incorporate commonality and variability; v) evaluate software architecture quality attributes.



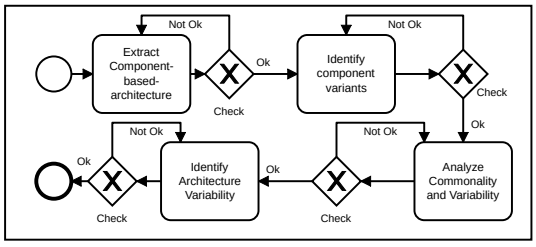
(a) BPMN Process Instantiated for [13]



(b) BPMN Process Instantiated for [1]



(c) BPMN Process Instantiated for [2]



(d) BPMN Process Instantiated for [36]

Figure 3: BPMN Representations of the Scenario Generated in SPLReePlan

from [8]. In this case, we had five activities: identify requirements on the architecture, identify commonalities and variabilities, restructure the architecture, incorporate commonality and variability, and evaluate software architecture quality attributes.

This scenario also used the dependency analysis retrieval technique, which appeared in the other three scenarios. Another similarity is the artifacts type, in this scenario architecture descriptions and source code were used, similar to the second and fourth scenarios.

5.2 Answering the RQs

With the analysis of the evaluation results, we were able to answer our RQs, as discussed in the following.

RQ1. How does SPLReePlan automate the planning of activities, artifacts, and techniques, for the SPL reengineering? Although we faced different challenges in all eight scenarios where our framework was applied, the results pointed out the SPLReePlan applicability in each individual scenario. In this sense, the SPLReePlan

1		**Phase:**	**Extract**
		Activity:	**Divide features with LSI**
		Description:	Use LSI to divide features and classes into common and variable partitions;
		Retrieval Technique:	Latent Semantic Indexing
	Artifact:1	**Input**	
		Name:	**Objected-oriented Source Code**
		Type	Development
		Description:	Souce code of the argoUML
		Extension:	.java
		Link (url):	https://github.com/argouml-tigris-org/argouml
		Last Update:	05-24-2020 by
	Artifact:3	**Output**	
		Name:	**Common and variable partitions**
		Type	Development
		Description:	Classes that implement common and optional features
		Extension:	.lsi
		Link (url):	https://github.com/argouml-tigris-org/argouml
		Last Update:	05-27-2020 by

(a) Report excerpt for scenario [13]

1		**Phase:**	**Extract**
		Activity:	**Extract raw architectural FM**
		Description:	Extract a raw architectural FM from a 150% architecture of the system
		Retrieval Technique:	Dependency Analysis
	Artifact:1	**Input**	
		Name:	**150% architecture of the system**
		Type	Architecture
		Description:	Composition of the architecture fragments of all the system plugins
		Extension:	.fm
		Link (url):	na
		Last Update:	07-23-2020 by User
	Artifact:2	**Output**	
		Name:	**architectural FM**
		Type	Architecture
		Description:	Considers both the software architect viewpoint and the variability actually supported by the system.
		Extension:	.fm
		Link (url):	na
		Last Update:	07-23-2020 by User

(b) Report excerpt for scenario [1]

Figure 4: BPMN Representations of the Scenario Generated in SPLReePlan

supported 100% of the artifacts (see Table 2), 100% of activities (considering minor changes), and approximately 85% of the techniques (17 out of 20).

As we replicated each original scenario as similar as possible, our main goal when analyzing this RQ was to understand the benefits of maintaining the aforementioned information. In this sense, we could see that such information is important for deciding which artifacts, techniques, and activities would be used, as well as for maintaining the reasons for making these decisions. This result is evidence by the reports generated in the tool, *e.g.*, Figures 4(a) and 4(b). These artifacts are important for the planning phase of SPLReePlan because they aid the users in abstract the techniques selection process by making them (users) consider different aspects of their context. Also, the reasons for making the decisions are important to understand why the decisions worked (or did not work) when performing the reengineering. Therefore, we can also argue that our framework provides a way of maintaining a repository of artifacts related to the reengineering. These artifacts can be used

Table 2: Results from the Evaluation.

Ref.	Art.		Tech.		Act.		Challenges							
	O	S	O	S	O	S	C1	C2	C3	C4	C5	C6	C7	C8
[13]	6	6	4	4	5	5	✓	✓	✓					
[1]	7	7	3	2	4	4	✓	✓	✓	✓				
[2]	5	5	3	3	3	3		✓	✓		✓	✓		
[36]	5	5	3	2	4	4			✓	✓			✓	
[4]	4	4	3	2	3	3			✓		✓			
[10]	4	4	1	1	4	4		✓						✓
[31]	7	7	2	2	6	6		✓						
[8]	9	9	1	1	5	5		✓	✓				✓	

O - Original Study; S - SPLReePlan Support.

when transforming the legacy system into an SPL, as well as when planning the SPL evolution.

RQ2. How does SPLReePlan allow the customization of the reengineering planning in different scenarios found in the literature?

As presented in Table 1, all eight scenarios were different, using a variety of different artifacts, techniques, and activities. Table 2 summarizes these results, showing how many artifacts, activities, and techniques from the original studies were supported by SPLReePlan. All artifacts used in the original studies could also be used in SPLReePlan, which shows that concerning artifacts flexibility, our framework handled more than 20 different types of artifacts. The same results apply to the activities instantiated, as all activities from the original studies could be instantiated into SPLReePlan generic process. Also, five different retrieval techniques were used, and their different combinations made all scenarios unique. This variety among the scenarios, alongside the fact that we could instantiate all the processes, gives us evidence suggesting that our framework is indeed customizable for different scenarios. However, we faced a few challenges. Some of these challenges directly impact the results of this RQ. These challenges were not crucial problems, however, they must be considered aiming at improving our proposal. For instance, some techniques from the original studies were not supported by SPLReePlan. Hence, these techniques could not be instantiated into the retrieval process. In addition, other challenges were identified during the evaluation. These challenges are discussed in the following section.

5.3 Challenges Faced and Lessons Learned

Table 3 summarizes the eight challenges identified during the evaluation. Also, it presents possible solutions to address them. The first challenge (C1) occurred in the first two scenarios. As the generic process does not support parallel activities, we had to handle this problem by transforming these activities into linear ones. However, a possible solution (PS1), would be to change the generic process and its instantiation to support parallel activities. The second challenge (C2) was the most frequent, appearing in six scenarios. This challenge occurred when an activity from the original study did not apply any retrieval technique. We argue that PS2 may address it by changing the framework to allow the users to assemble activities in the generic process without the need of assembling retrieval techniques for them. Challenge C3 was related to some phases of the generic process not receiving activities during the assembly. For instance, in the second scenario, the *group* activity was not

Table 3: Challenges Identified During the Evaluation.

ID	Description	ID	Possible Solution
C1	Some activities from the original study were performed in parallel	PS1	Change the generic process to support parallelism
C2	An activity did not apply any retrieval technique	PS2	Allow the users to define activities without the application of retrieval techniques
C3	Some phases of the generic process did not have an activity related to it	PS3	Change the generic process to make activities or-optional
C4	A technique that was not supported by SPLReePlan was used in an activity	PS4	Allow the user to add new techniques into the framework
C5	One activity used more than one retrieval technique	PS5	Allow the user to assemble multiple retrieval techniques into an activity
C6	One activity was part of more than one phase of the generic process	PS6	Allow the user to create activities that are part of more than one phase
C7	The approach did not aim at generating a FM	PS7	Change the last activity to aim at creating any kind of variability model
C8	The original scenario had no input artifacts	PS8	Make the inclusion of technical artifacts optional during the initiation step

used. As defined in its BPMN representation, however, all its phases are mandatory. Changing the generic process to make its activities Or-optional, meaning that at least one is mandatory is a possible solution (PS3). Challenge C4 was faced twice when techniques that were not present in the framework were used. We believe that C4 can be addressed by allowing the users to add or suggest the addition of new retrieval techniques into our framework (PS4). This could be done by providing a template or an online form for users to fill and send us.

Challenge C5 was identified when one of the activities from the original study applied more than one retrieval technique in parallel. As our framework does not give support to this, we should allow users to assemble multiple retrieval techniques into a single activity (PS5). The next challenge, C6, is related to one activity being part of more than one phase of the generic process, for instance, extract and categorize. The possible solution (PS6) would be to allow the users to create activities that are part of more than one phase. The seventh challenge identified (C7) was faced when two of the scenarios did not aim at generating an FM at the end of their process. This challenge is different because SPLReePlan's final output should be the FM.

However, we still understand that C7 is an opportunity for improvement as a company that works or desires to have a variability artifact different from an FM, such as an SPL architecture. Thus, we plan to change the last activity of the process to support the creation of different variability models (PS7). Challenge C8 appeared only in one scenario [10], where their approach did not have input-only technical artifacts. Input-only artifacts are those collected and analyzed during the SPLReePlan's initiation step , which can be organizational or technical. To address C8, we would make the inclusion of technical artifacts optional during the initiation step (PS8), relying on the decision only on organizational artifacts.

By analyzing these challenges to propose possible solutions we identified at least eight points for improving SPLReePlan. However, we can also argue that these results can contribute to other researchers/practitioners that are developing or applying similar proposals/tools considering different scenarios. As most of the challenges faced were related to the differences across the scenarios used as input for the evaluation, such challenges can occur when using similar tools and proposals. Thus, the possible solutions identified can also be used to address these challenges. However, we should plan and conduct an additional evaluation considering similar tools to analyze if these challenges persist when applying them to different scenarios.

6 THREATS TO VALIDITY

Next, we present the threats of our study and how we tried to mitigate them [38].

Construct validity: an important threat concerns the selection of relevant scenarios for conducting the customization. This may be a threat if the scenarios selected were not reliable enough to be considered relevant when extracting and analyzing the results of the evaluation. To mitigate this, we used a well-known catalog of SPL case studies [28]. Another threat is related to the selection of scenarios that may or may not be useful for our evaluation, as they may not possess the information we require. To mitigate this problem, we defined and applied three inclusion criteria when selecting the studies from where the scenarios would be extracted.

Internal Validity: a possible threat is related to errors when conducting and documenting the results of the evaluation. This may lead to erroneous conclusions, resulting in misleading answers for the RQs. To mitigate this problem, we clearly defined four steps for conducting the evaluation. Also, we conducted and published a pilot execution [26] to evaluate how the steps performed helped to answer the RQ.

External Validity: concerning the relevance of our findings to other studies, we established a replicable protocol, using as input studies found in a public and well-known catalog of case studies. Also, we provide all artifacts from the evaluation in an open data repository [25], allowing reproducibility.

Reliability: a possible threat is related to problems with the data dependency caused by the researchers conducting the evaluation. To mitigate this problem, we defined four different metrics for the RQs, describing how they may be answered by analyzing the results. An additional threat is the number of scenarios selected. Despite of having only eight scenarios, we argue that they allowed us properly investigating the flexibility and limitations. However, we understand that a larger dataset could complement our results.

7 RELATED WORK

Studies on feature retrieval: studies proposing a process for feature retrieval in SPL reengineering usually focus on technical aspects and few artifacts, namely requirements, design, and source code [5]. Requirements artifacts are used in several studies [1, 6, 29], being one of the most common artifacts amongst the approaches as mapped in [5]. Design models are used as input in [1, 22], while source code is used in [15, 20, 29]. Source code is the most used artifact for retrieving features. Domain information, however, is only used in a few studies [14]. This type of information may be more

related to organizational than technical aspects, corroborating with our argument that such information is usually not considered when planning the reengineering. Although these approaches use different types of artifacts, their flexibility is still limited. For instance, in [6] many artifacts are mandatory, hindering the customization of several scenarios. The approach presented in [29], however, was designed to be generic regarding the artifact types, allowing the integration of new artifact types. However, this approach does not consider organizational aspects by default, as done in SPLReePlan.

Customization for different scenarios: the customization considering different scenarios and contexts allows the adaptability of the approaches considering different organizational scenarios. This customization is present in approaches such as PuLSE-Eco [35], which is part of the PuLSE framework [11], supporting SPL scoping. Another work presented in [3], defines a set of rules for different scenarios, handling scoping costs according to the current scenario, thus, making their approach customizable. In comparison, SPLReePlan offers a generic feature retrieval process as well as specific activities to collect organizational information.

Tool supporting: although we may find different tools to support feature retrieval, most of them focus only on technical information [18, 30]. This limits the industrial adoptability of the tool as it may not be adaptable to different scenarios where other levels of information are used. Thus, when defining SPLReePlan supporting tool, in addition to technical information, we also considered organizational aspects. Also, we understand that several tools are only developed as prototypes or proof of concept [17, 33], not considering end-user requirements during their development. This limits the applicability of the tools in real scenarios since usability and scalability are important properties taken into account by practitioners when deciding whether to use them or not.

Comparison among SPLReePlan, PAxSPL, and Assunção *et. al.* process: Figure 5 presents a comparison among SPLReePlan with PAxSPL and the process defined by Assunção *et. al.* [5]. As shown, the *initiation* and *instantiation* steps correspond to PAxSPL's *prepare* and *assemble* phases, respectively. Both phases occur before the reengineering process reported in [5]. The main goal of these steps is to plan and define the process for reengineering the features from the legacy system. The main difference between SPLReePlan and PAxSPL in these phases is that SPLReePlan automates them with a supporting tool. The SPLReePlan's *documentation* step is related to the *execute* phase from PAxSPL. This step is also related to the *detection* step from Assunção *et al.* [5]. In this step, the goal is to apply the planned process (also called "assembled process") to detect and retrieve the features from the legacy system. SPLReePlan differentiates from PAxSPL by implementing the heuristics that can be used for recommending more suitable techniques based on the documentation gathered about the reengineering scenario. It is important to mention that this is only possible because the tool supports the registration of this documentation in the first two steps. SPLReePlan's *analysis* step is also related to the *execute* phase from PAxSPL, as well as the *analysis* from Assunção *et al.*. The goal here is to analyze the features detected, group, categorize them, and generate a variability model, *e.g.*, feature model.

SPLReePlan gives support for creating this model using the features documented in the previous step. When creating this model,

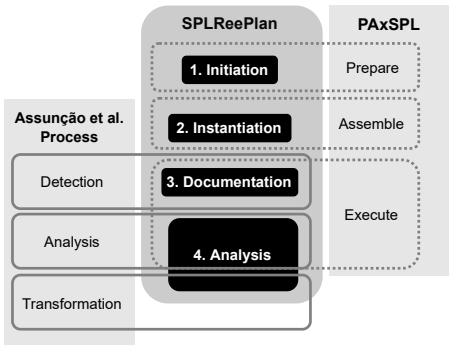


Figure 5: Comparison among SPLReePlan, PAxSPL and Assunção *et. al.* process [5]

it is also possible to include a reference to the artifacts that are related to each feature. This allows SPLReePlan *analysis* step to go beyond PAxSPL's *execute* phase into the *transformation* from the reengineering process. This is done as the SPLReePlan framework supports the generation of product configurations from the variability model created. As mentioned, all steps of the SPLReePlan framework are automated by a supporting tool. This automated support, can aid the SPL reengineering planning process and motivates companies to conduct the extractive adoption of SPLs.

8 CONCLUSION

The SPL reengineering process is adopted by organizations that aim to migrate their legacy software into a technology focusing on systematic reuse. Organizational scenarios impact the SPL reengineering process as scenario variables can determine which retrieval techniques are more suited to them. In this sense, a process customized based on these variables may be a satisfactory solution for reducing this impact. In addition, when searching for approaches and methods that give support to this process considering these variables, we could only find PAxSPL. However, PAxSPL presented limitations identified as results of a case study conducted [27], as well as not being automated by any tool.

Therefore, we propose a framework called SPLReePlan that gives support for users to plan the SPL reengineering by customizing their feature retrieval processes. Our framework was constructed by considering the SPL reengineering planning activities, limitations of works in the field, and requirements of end-user tools. As the results of our evaluation pointed out, SPLReePlan can be applied in different scenarios considering a variety of scenario variables such as artifacts, techniques, and activities. Although some challenges were faced, we can argue that our framework mitigates some problems that practitioners and researchers may find when adopting reengineering approaches to a specific scenario, especially if this scenario requires flexibility or customization. Also, with the artifacts being registered in the tool, users can use it as a repository for product documentation. This repository can be used for different reasons, including the analysis of the features for transforming the legacy software into SPL as well as planning its evolution. Besides, as our tool was developed considering end-user requirements, it

intends to be applied in real scenarios, differently from prototype or proof-of-concept tools.

For future work, we intent to extend SPLReePlan to include additional feature retrieval techniques found in the literature. In addition, we plan to interview the authors of the eight scenarios used in the evaluation to collect and analyze their opinion concerning the instantiation of the scenarios. We also plan to conduct a case study in a real-world scenario, in partnership with companies.

ACKNOWLEDGMENTS

This work was partially funded by Unipampa 10/2019 - PAPG/2019, CNPq grant no. 408356/2018-9, and FAPERJ PDR-10 program grant no. 202073/2020.

REFERENCES

[1] M. Acher, A. Cleve, P. Collet, P. Merle, L. Duchien, and P. Lahire. 2013. Extraction and evolution of architectural variability models in plugin-based systems. *Software & Systems Modeling* 13, 4 (2013), 1367–1394.

[2] R. Al-Msie'Deen, D. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. Eyal-Salman. 2012. An approach to recover feature models from object-oriented source code. *Actes de la Journée Lignes de Produits* (2012), 15–26.

[3] H.I. Alsawalqah, S. Kang, and J. Lee. 2014. A method to optimize the scope of a software product platform based on end-user features. *Journal of Systems and Software* 98 (2014), 79 – 106. https://doi.org/10.1016/j.jss.2014.08.034

[4] V. Alves, C. Schwanninger, L. Barbosa, A. Rashid, P. Sawyer, P. Rayson, C. Pohl, and A. Rummler. 2008. An exploratory study of information retrieval techniques in domain analysis. In *12th International Software Product Line Conference*. IEEE, 67–76.

[5] W. Assunção, R. Lopez-Herrejon, L. Linsbauer, S. Vergilio, and A. Egyed. 2017. Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Software Engineering* (2017), 1–45.

[6] G. Bécan, M. Acher, B. Baudry, and S. Nasr. 2013. Breathing ontological knowledge into feature model management. Available at: https://hal.inria.fr/hal-00874867/. Access in: 26 april 2020.

[7] J. Bosch. 2006. The challenges of broadening the scope of software product families. *Commun. ACM* 49, 12 (2006), 41–44.

[8] H. P. Breivold, S. Larsson, and R. Land. 2008. Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies. In *2008 34th Euromicro Conference Software Engineering and Advanced Applications*. 232–239.

[9] S. Chardigny, A. Seriai, M. Oussalah, and D. Tamzalit. 2008. Extraction of Component-Based Architecture from Object-Oriented Systems. In *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*. 285–288.

[10] K. Chen, W. Zhang, H. Zhao, and H. Mei. 2005. An approach to constructing feature models based on requirements clustering. In *Requirements Engineering, 2005. Proceedings. 13th IEEE International Conference on*. IEEE, 31–40.

[11] J. deBaud and K. Schmid. 1999. A systematic approach to derive the scope of software product lines. In *Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No.99CB37002)*. 34–43. https://doi.org/10.1145/302405.302409

[12] S. Dumais. 2004. Latent semantic analysis. *Annual review of information science and technology* 38, 1 (2004), 188–230.

[13] H. Eyal-Salman, D. Seriai, and C. Dony. 2013. Feature-to-code traceability in a collection of software variants: Combining formal concept analysis and information retrieval. In *14th International Conference on Information Reuse and Integration*. IEEE, 209–216.

[14] A. Ferrari, G. O. Spagnolo, and F. Dell'Orletta. 2013. Mining commonalities and variabilities from natural language documents. In *17th International Software Product Line Conference*. ACM, 116–120.

[15] S. Fischer, L. Linsbauer, R. E. Lopez-Herrejon, and A. Egyed. 2014. Enhancing clone-and-own with systematic reuse for developing software variants. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE, 391–400.

[16] B. Ganter and R. Wille. 2012. *Formal concept analysis: mathematical foundations*. Springer Science & Business Media.

[17] E. N. Haslinger, R. E. Lopez-Herrejon, and A. Egyed. 2011. Reverse engineering feature models from programs' feature sets. In *18th Working Conference on Reverse Engineering*. IEEE, 308–312.

[18] Florian Heidenreich, Jan Kopcsek, and Christian Wende. 2008. FeatureMapper: mapping features to models. In *Companion of the 30th international conference on Software engineering*. ACM, 943–944.

[19] A. Hubaux, P. Heymans, and D. Benavides. 2008. Variability modeling challenges from the trenches of an open source product line re-engineering project. In *12th International Software Product Line Conference*. IEEE, 55–64.

[20] K. Kang, M. Kim, J. Lee, and B. Kim. 2005. Feature-oriented re-engineering of legacy systems into product line assets–a case study. In *International Conference on Software Product Lines*. Springer, 45–56.

[21] J. Krüger, W. Mahmood, and T. Berger. 2020. Promote-Pl: A Round-Trip Engineering Process Model for Adopting and Evolving Product Lines. In *24th ACM Conference on Systems and Software Product Line: Volume A* (Montreal, Quebec, Canada) *(SPLC '20)*. ACM, New York, NY, USA, Article 2, 12 pages. https://doi.org/10.1145/3382025.3414970

[22] U. Kulesza, V. Alves, A. Garcia, A. Neto, E. Cirilo, C. De Lucena, and P. Borba. 2007. Mapping features to aspects: A model-based generative approach. In *Early Aspects Workshop*. Springer, 155–174.

[23] M. A Laguna and Y. 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.

[24] L. B. Lisboa, V. C. Garcia, D. Lucrédio, E. S. de Almeida, S. R. de Lemos Meira, and R. P. de Mattos Fortes. 2010. A systematic review of domain analysis tools. *Information and Software Technology* 52, 1 (2010), 1 – 13. https://doi.org/10.1016/j.infsof.2009.05.001

[25] L. Marchezan, W. KG Assunção, J. Carbonell, E. M. Rodrigues, M. Bernardino, and F. Basso. 2021. *SPLReePlan Data set*. https://doi.org/10.5281/zenodo.5091627

[26] L. Marchezan, J. Carbonell, E. Rodrigues, M. Bernardino, F. P. Basso, and W. K. G. Assunção. 2020. Enhancing the Feature Retrieval Process with Scoping and Tool Support: PAxSPL_v2. In *Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B* (Montreal, QC, Canada) *(SPLC '20)*. Association for Computing Machinery, New York, NY, USA, 29–36. https://doi.org/10.1145/3382026.3425767

[27] L. Marchezan, E. Rodrigues, M. Bernardino, and F. P. Basso. 2019. PAxSPL: A feature retrieval process for software product line reengineering. *Software: Practice and Experience* 49, 8 (2019), 1278–1306. https://doi.org/10.1002/spe.2707

[28] J. Martinez, W. K. G. Assunção, and T. Ziadi. 2017. ESPLA: A Catalog of Extractive SPL Adoption Case Studies. In *Proceedings of the 21st International Systems and Software Product Line Conference - Volume B* (Sevilla, Spain) *(SPLC '17)*. ACM, New York, NY, USA, 38–41. https://doi.org/10.1145/3109729.3109748

[29] Jabier Martinez, Tewfik Ziadi, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Bottom-up adoption of software product lines: a generic and extensible approach. In *Proceedings of the 19th International Conference on Software Product Line*. ACM, 101–110.

[30] C. Nunes, A. Garcia, C. Lucena, and J. Lee. 2012. History-sensitive heuristics for recovery of features in code of evolving program families. In *16th International Software Product Line Conference*. ACM, 136–145.

[31] P. Paškevičius, R. Damaševičius, E. Karčiauskas, and R. Marcinkevičius. 2012. Automatic Extraction of Features and Generation of Feature Models from Java Programs. *Information Technology And Control* 41, 4 (2012), 376–384.

[32] J. A. Pereira, K. Constantino, and E. Figueiredo. 2015. A systematic literature review of software product line management tools. In *International Conference on Software Reuse*. Springer, 73–89.

[33] J. Rubin and M. Chechik. 2010. From Products to Product Lines Using Model Matching and Refactoring.. In *SPLC Workshops*. 155–162.

[34] G. Salton, A. Wong, and C. Yang. 1975. A vector space model for automatic indexing. *Commun. ACM* 18, 11 (1975), 613–620.

[35] K. Schmid. 2002. A Comprehensive Product Line Scoping Approach and Its Validation. In *Proceedings of the 24th International Conference on Software Engineering* (Orlando, Florida) *(ICSE '02)*. ACM, New York, NY, USA, 593–603.

[36] A. Shatnawi, A. Seriai, and H. Sahraoui. 2014. Recovering Architectural Variability of a Family of Product Variants. In *Software Reuse for Dynamic Systems in the Cloud and Beyond*. Springer International Publishing, Cham, 17–33.

[37] F. Van der Linden, K. Schmid, and E. Rommes. 2007. *Software product lines in action: the best industrial practice in product line engineering*. Springer Science & Business Media.

[38] C. Wohlin, P. Runeson, M. Höst, M. C Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in software engineering*. Vol. 1. Springer Science & Business Media.