


RESEARCH

Open Access



# Product line architecture recovery with outlier filtering in software families: the Apo-Games case study

Crescencio Lima<sup>1,2\*</sup> , Wesley KG Assunção<sup>3</sup>, Jabier Martinez<sup>4</sup>, William Mendonça<sup>3</sup>, Ivan C Machado<sup>1</sup> and Christina FG Chavez<sup>1</sup>

## Abstract

Software product line (SPL) approach has been widely adopted to achieve systematic reuse in families of software products. Despite its benefits, developing an SPL from scratch requires high up-front investment. Because of that, organizations commonly create product variants with opportunistic reuse approaches (e.g., copy-and-paste or clone-and-own). However, maintenance and evolution of a large number of product variants is a challenging task. In this context, a family of products developed opportunistically is a good starting point to adopt SPLs, known as extractive approach for SPL adoption. One of the initial phases of the extractive approach is the recovery and definition of a product line architecture (PLA) based on existing software variants, to support variant derivation and also to allow the customization according to customers' needs. The problem of defining a PLA from existing system variants is that some variants can become highly unrelated to their predecessors, known as outlier variants. The inclusion of outlier variants in the PLA recovery leads to additional effort and noise in the common structure and complicates architectural decisions. In this work, we present an automatic approach to identify and filter outlier variants during the recovery and definition of PLAs. Our approach identifies the minimum subset of cross-product architectural information for an effective PLA recovery. To evaluate our approach, we focus on real-world variants of the Apo-Games family. We recover a PLA taking as input 34 Apo-Game variants developed by using opportunistic reuse. The results provided evidence that our automatic approach is able to identify and filter outlier variants, allowing to eliminate exclusive packages and classes without removing the whole variant. We consider that the recovered PLA can help domain experts to take informed decisions to support SPL adoption.

**Keywords:** Software product lines, Product line architecture, Variability, Product line architecture recovery

## Introduction

Software product line (SPL) is a widely adopted approach for developing and managing a family of software products. SPLs leverage systematic software reuse, since a set of products, designed for a specific domain or market segment, share common parts [1]. By adopting SPLs, companies achieve benefits such as reduced time-to-market, planned delivery of products, homogenization of the quality of the products, easier maintenance, and evolution of

variants while meeting specific customer or market needs [2, 3].

Despite its benefits, SPL adoption requires high up-front investments, the return-on-investment is usually conditioned to a deep knowledge of the domain/market segment, and its evolution in the mid- and long-term. This also makes SPLs more suitable for mature domains where the variability of the systems is well established. However, this is not the scenario where the great majority of software companies are included. Usually, many companies rely on less sophisticated reuse approaches, focusing on immediate needs of their customers, without taking into account systematic reuse with long-term vision.

These less sophisticated reuse approaches are collectively called *opportunistic reuse*, or ad hoc reuse [4, 5],

\*Correspondence: [crescencio@gmail.com](mailto:crescencio@gmail.com)

<sup>1</sup>Federal University of Bahia, Ademar de Barros - Campus de Ondina, Salvador, 40170-110, Brazil

Full list of author information is available at the end of the article

that includes techniques such as clone-and-own or copy-and-paste reuse. Opportunistic reuse provides short-term benefits; however, once the design quality is not considered as a factor during or after reuse, it results in extensive refactoring and contributes to maintenance owes, leading to unanticipated behavior, violated constraints, conflict in assumption, fragile structure, and software bloat [6].

Due to the benefits of systematic reuse and because of the problems of opportunistic reuse, re-engineering of existing products to SPLs has been receiving attention from industry and academia [7, 8]. An industrial survey pointed that around 50% of the companies that adopt SPLs do not start from scratch, but rather start with a set of existing variants [9].

One challenge on re-engineering existing system variants into SPLs is that, by using the opportunistic reuse, some variants become highly unrelated to their predecessors. Because of the need of a significant modification in the variant to accommodate a specific functionality, or the inclusion of new developers in the development process, certain variants are transformed in outliers [10]. Outliers certainly fulfill their purpose of providing customized functionalities for a specific scenario; however, since they are maintained and evolved as part of a family of products, outliers become problematic because of erroneous relations might be identified, more detailed knowledge of the implementation might fade away, or the expert in the variant may leave the development team. Furthermore, analogously to the architecture of single systems, *drift* and *erosion* [11] can happen in a family of products as a whole, making more difficult the re-engineering of the variants into SPLs.

To develop an SPL, a fundamental step is to define the product line architecture (PLA). PLA represents the architecture of a family of products describing common and variable components of the SPL. With the goal of re-engineering existing products into SPLs, one of the initial steps is to recover the architecture considering the family of products. While architecture recovery techniques have been largely studied for single systems [12], recovery techniques for families of systems have been overlooked [8] despite their importance in SPL adoption.

During the recovery of a PLA from existing products, the analysis and integration of outliers will demand additional effort for architects and engineers. Each outlier includes too much noise in the common structure, requires specific analysis, and complicates architectural decisions. To overcome this problem, in a previous work, we present a cost-effective PLA recovery strategy that identifies and filters outliers [13]. The main characteristics of our previous work are as follows: (i) it is an automatic approach for the identification of the minimum subset of cross-product architectural information for an effective PLA recovery. The assumption is that efficient

and comprehensive PLAs obtained through architecture recovery can be automated by “pruning” the set of variants’ architectures, and (ii) in our previous evaluation, we took into account real-world variants created by means of clone-and-own strategy. Concretely, we used 20 open-source variants of medium-size Java games known as Apo-Games in the SPL research community [14].

In this study, we extend our previous work [13] as follows:

- Further details of the proposed approach are described.
- An illustrative example considering a real-world set of system variants is presented.
- We introduce a guideline to support of the PLA recovering process.
- Inclusion of 14 Apo-Games new variants (nine desktops and five mobiles) in addition to the 20 ones of our previous study.
- An in-depth analysis of the results is presented.

Our approach supports the automation of the PLA recovery and the identification of outlier variants. We propose the improvement of the recovered PLA by combining threshold analysis with outlier filtering and metric analysis. The results obtained with the Apo-Games case study reveal that a real-world project has outlier variants, there is a correlation between the size of the variant and the existence of exclusive implementation elements, and the filtering of outliers has a positive impact on the recovered PLA, mainly in the high level of abstraction, namely in package representation. Furthermore, the analysis of desktop and mobile variants of the same game allows identification of variability and commonalities in implementations for different platforms.

The remainder of the paper is organized in sections that present background information (the “[Background](#)” section), describe our approach to PLA recovery (the “[Proposed approach](#)” section), the experimental study conducted (the “[Study design](#)” section), and the study execution and analysis (the “[Results](#)” section), provide interpretation and answers to research questions (the “[Discussion](#)” section), discuss related work (the “[Related work](#)” section), and present concluding remarks and recommendations for future work (the “[Concluding remarks](#)” section).

## Background

In this section, we provide background and definitions needed for the rest of the paper.

### Apo-Games projects

The Apo-Games<sup>1</sup> is a set of medium-sized games that have been implemented based on the clone-and-own approach. The Apo-Games has been proposed as a

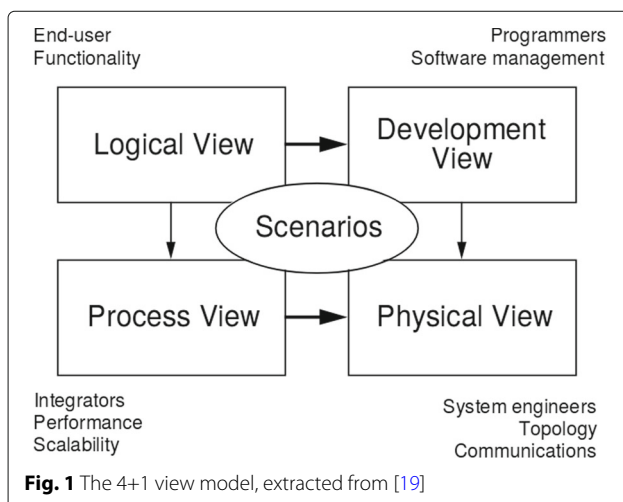
good candidate for research in the context of reverse engineering of variability [14]. Apo-Games evolved over time because of the inclusion of new games and adaptations. These games are composed of Java desktop and Android applications (Android applications are Java-based programs targeting mobile devices). Java desktop has evolved since 2007 until 2014. In 2012, the development of Android games started. In this work, we deal with the Java games of the Apo-Games repository<sup>2</sup> [14]. The source code for each game ranges from 2.5 KLOC to 46.7 KLOC. These games are medium-sized projects and together sum up to an overall size of 178.2 KLOC. In addition, we also deal with variants available as Jar files. Further details of Apo-Games variants are described in the section “Study design.”

### Product line architecture

Variability at the architectural level can be a complex concept that should be addressed across all the software life cycle [15]. So far, architectural variability has been addressed in the SPL domain [15, 16]. To address variability in SPL projects, the notion of PLA [17] was introduced to capture the core design of all products including commonalities and variability of several product instances [18]. In other words, PLA is a special type of architecture that describes commonality (core assets) and variability (varying assets), so it plays a crucial role to develop families of products as it is the basis for the architectures of all SPL products within the family [2]. At the same time, PLA supports the variability management in the design process allowing high level abstraction for its comprehension and common understanding at the organizational level [11].

### Architecture views

Kruchten [19] proposed the 4+1 architectural model to represent software architecture from different viewpoints.



The model is organized in five views, as depicted in Fig. 1: logical, development, process, physical, and the fifth view combines the former ones as a means to illustrate and explain the overall architecture.

Despite the importance of software architecture in the context of software development, the variants created by using opportunistic reuse seldom provide formal and up-to-date architectural representations that explicitly show system organization. Such information is only present in implementation artifacts, i.e., source code.

The Apo-Games variants we deal with in this work fit the aforementioned scenario (absence of explicit representations of the architecture apart from the development view). Since the extracting process is based on the games’ source code, we focus on the development view of the 4+1 architectural model for representing the recovered PLA and identifying the architectural variability. The development view, at the top right corner of Fig. 1, is a model that represents software components as collections of source code artifacts (e.g., classes and packages) and software connectors as relations between these components (e.g., imports, calls, uses).

### Architecture recovery

Software architecture recovery (SAR) processes are organized into three main categories [20]: (i) bottom-up, start with low-level knowledge to recover architecture; (ii) top-down, start with high-level knowledge aiming to discover the architecture; and (iii) hybrid, combining the previous two.

In this paper, we use bottom-up processes to recover PLAs. We create a source code model (low-level knowledge) for the PLA. Then, we raise the abstraction level, by providing a representation for the PLA using the development views.

### PLA metrics

PLA metrics aims to assess certain qualities of the PLA. In this work, we decided to use four metrics: SSC and SVC (taken from the PLA metrics proposed by Zhang et al. [21] focusing on structural components), and RSC and RVC (similar to the previous ones but focusing on the relations between components). Their definitions are provided below.

Structure similarity coefficient (SSC) is used to measure the similarity between PLA components, while structure variability coefficient (SVC) is used to measure the structure variability of the PLA. Given  $C_c$ , the number of common components in the PLA, and  $C_v$ , the number of variable components, SSC and SVC are defined as follows.

$$SSC = \frac{|C_c|}{|C_c| + |C_v|} \quad SVC = \frac{|C_v|}{|C_c| + |C_v|}$$

Relation similarity coefficient (RSC) is used to measure the similarity between PLA relations, and relation variability coefficient (RVC) measures the variability of PLA relations. Given  $R_c$ , the number of common relations in the PLA, and  $R_v$ , the number of variable relations, RSC and RVC are defined as follows.

$$RSC = \frac{|R_c|}{|R_c| + |R_v|} \quad RVC = \frac{|R_v|}{|R_c| + |R_v|}$$

The SSC and SVC metrics are highly related given that the sum of SSC and SVC will be always 1. Values close to 1 for SSC means that there are few optional components, and values close to 0 means that the PLA of the different variants does not have many components in common. We can draw similar conclusions regarding the RSC and RVC metrics because the same relationship happens between them.

### Proposed approach

Figure 2 presents the proposed automatic PLA recovery approach. Our approach has generic steps and can be implemented to work on different contexts. In the implementation presented in this work, we deal with Java projects, then some specific tools were used for this context.

First, in ①, we select the input variants source code. Examples of input candidates are SPL products, systems from the same domain, and projects implemented using clone-and-own strategy. Then, in the information extraction ②, we extract the structural information of each variant. Our implementation relies on the Stan4<sup>3</sup> tool, but for other contexts, different tools that collect structural information can be used. To allow the identification of the common and variable packages and classes,

in ③, we pre-process information of variants to eliminate nomenclatures or prefixes specific for a single game variant. For instance, in the ApoBot variant, we changed ApoBotPlayer to Player because it allows us to identify the variants' common components. These preparatory refactorings were already proposed in the literature as a way to align variants and facilitate the migration [22].

As a result, in ④, we export the extracted information in Trivial Graph Format (TGF) files [23]. We use these files as an input for the variability identification ⑤. This step is composed of two sub-tasks, namely threshold analysis ⑥ and formal sub-analysis ⑦, designed to identify and filter the outliers. For the threshold analysis ⑥, the goal is to provide the reduction of exclusive components (packages or classes) without eliminating the whole outlier variants (variants with a considerate number of exclusive components). For this step, we extended the PLAR tool [24] to eliminate the components below a defined threshold value. Formal concept analysis (FCA) ⑦ generates an auxiliary representation to support better understanding of the implementation among variants. In our context, this representation is obtained with the Bottom-Up Technologies for Reuse (BUT4Reuse)<sup>4</sup> tool [25].

In ⑧, we present the outputs of the PLA recovery approach: metrics, report, design structure matrix (DSM), development view (packages and classes), and concept lattice.

Figure 3 shows the PLA recovery inputs and outputs organized in three layers. At the bottom layer (layer 0), the source code provided as input allows the extraction of structural information. As outputs of the PLA recovery approach, the middle layer (layer 1) raised the abstraction level using the information of classes, and the top layer (layer 2) gathered the classes in packages. In layer 2, we provided one consolidated package diagram of the PLA.

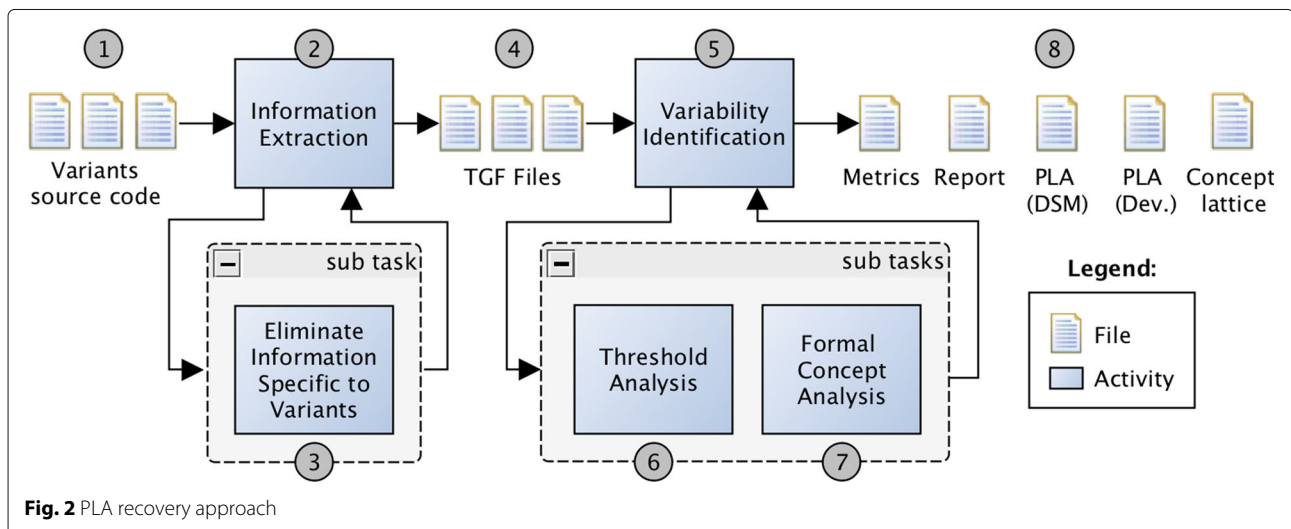
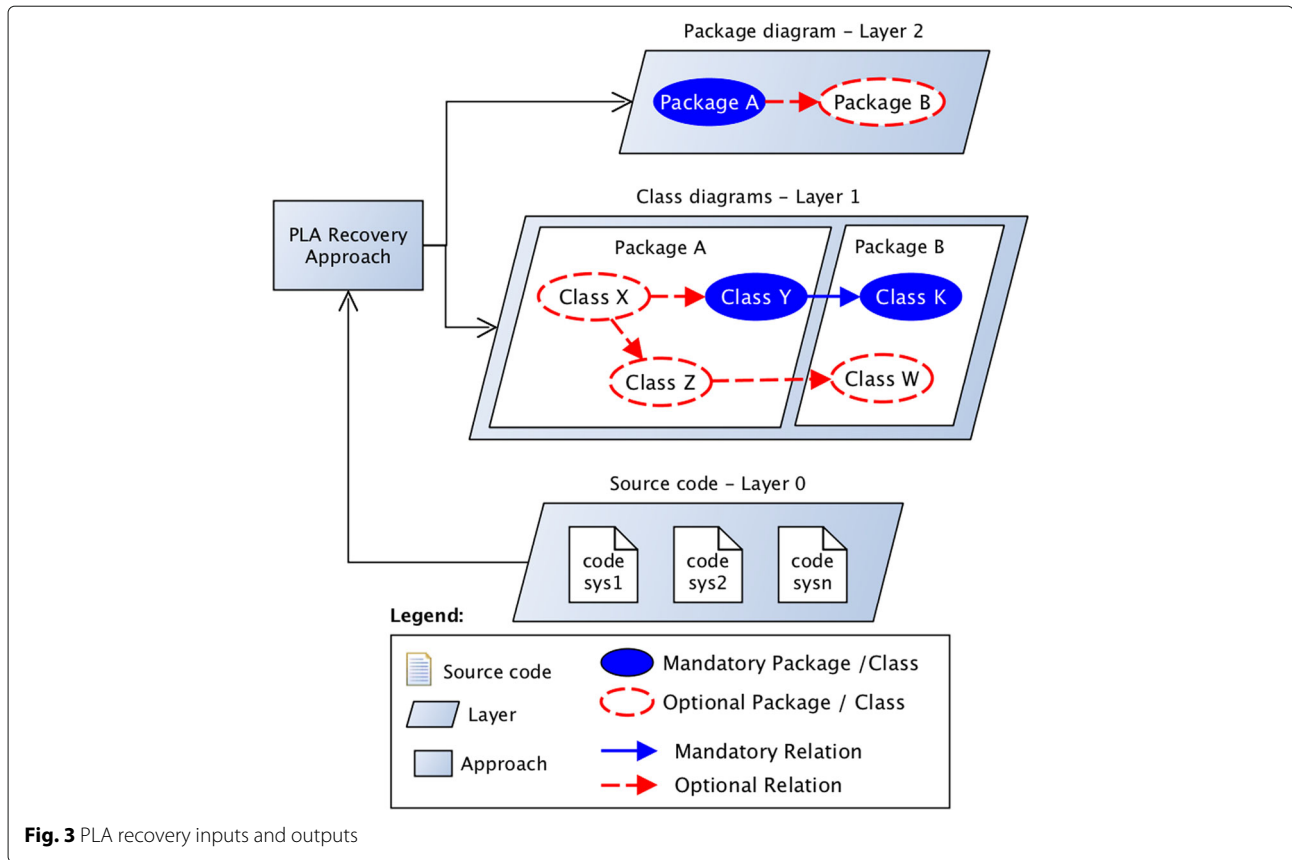


Fig. 2 PLA recovery approach



**Fig. 3** PLA recovery inputs and outputs

Then, in layer 1, we created a consolidated class diagram for each package identified in layer 2. For instance, package A is a mandatory package because it appears in every Apo-Games project (layer 2). In layer 1, the developer can check the classes that implemented package A.

Despite of our approach being designed to deal with possible problems existing in variants, such as the corrections of names in implementation of specific to variants, the quality of the recovered PLA is based on the quality of source code input. For example, similar yet different assets might lead the approach to generate inappropriate results.

**PLA recovery guideline**

In a previous study, we identified the lack of guidelines to support practitioners on using existing tools to conduct PLA recovery [26]. Based on the evidence obtained in our studies [13, 26–28], we define a guideline to help practitioners on using our approach.

Table 1 presents the recovery guideline that documents steps and rationale for recovering a PLA from a set of variants developed with the clone-and-own strategy. We describe the *problem* and discuss a possible *solution*. Then, we define the *preconditions* to start the PLA recovery. We mapped the *steps* to perform the recovery approach, and the results are presented in the *post*

*conditions*. We also provided some *hints* for improving the PLA recovery, and *trade-offs* are designed to deal with outlier variants.

**Motivating example**

The goal of this motivating example is to illustrate our PLA recovery approach. Next, we present five Android Apo-Games variants. We selected these five projects because they are relatively small for illustrative purposes. The approach steps are presented in the following subsections.

**Variants’ analysis and extraction**

We extracted the variants’ architecture using Stan4J tool. Figure 4 shows the extracted architecture of the five Apo-Games variants—(a) ApoClock, (b) ApoSnake, (c) ApoMono, (d) ApoDice, and (e) myTreasure. They are mobile games developed for the Android devices.

Figure 4a presents the packages and relationships of the ApoClock game. The numbers in the relationships describe the level of dependency between the packages. For instance, there are five relationships between .apoclock.game and .apoclock.editor, which means that .apoclock.game calls five classes from .apoclock.editor. This notion is the same used in other variants.



**Table 1** PLA recovery guideline*PLA recovery guideline*

*Intent:* recover the PLA from a set of variants

*Problem*

It is not uncommon for software companies to adopt SPL using a *clone-and-own* strategy, by copying, adding, or removing functions from existing products [29]. This approach leads to ad hoc product portfolios of multiple yet similar variants [30]. With the growth of products' portfolio, the management of variability and reuse becomes more complex [31]. A PLA for the SPL could be recovered from its variants and be used to tame complexity and drive SPL evolution.

However, SAR for SPL requires additional effort to identify the variability spread on several implemented variants and represent them at the architectural level. In this context, we may ask:

*How do we recover a software architecture that unveils variability and commonality for such a portfolio of clone-and-own related variants?*

This problem is difficult because:

- Variants may be large and complex.
- Each cloned variant may have evolved independently from others occasionally becoming an outlier.

Yet, solving this problem is feasible because:

- There are many SAR techniques for single systems.
- You have the source code of a set of variants.
- Good design practices promote the implementation of well-modularized units.

*Solution*

Application of the PLA recovery approach in set of variants with the support of the PLAR tool [24].

*Preconditions*

1. Set of variants' source code developed using clone-and-own strategy available.

*Steps*

1. Get the variants' source code.
2. Select an extraction tool to recover the variants' structural information.
3. Select an adapter to verify the elements names eliminating information specific to a variant.
4. Perform the variability identification using the PLAR tool support to automate the process.
5. Analyze the collected metrics, report, and visualization of the recovered PLA.
6. Run the threshold analysis according to the specified threshold value.
7. [optional]—Go back to step 5 until you reach the desired metric values.

*Post conditions*

1. Set of reports, metrics, design structure matrices (DSMs), development views, and UML diagrams from the recovered PLAs according to threshold values.

*Hints for improving the PLA recovery*

- Use the report to identify the elements' frequency.
- Perform threshold analysis according to the elements' frequency.

*Trade-offs*

- The higher the threshold value, the higher will be the elimination of variable elements.

**Application of the PLA recovery**

Figure 5 shows the development view (one of the outputs) of the PLA recovery approach. Moreover, we identified the variability in the architectural level. In this figure, the recovered PLA is represented in packages, where the game, entity, and main are mandatory packages (blue-filled ellipses) because they are implemented in all the five variants. On the other hand, editor, highscore, highscore, userlevels, and level are optional (red-dashed ellipses) because they are implemented by only some variants.

Figure 6 presents the recovered design structure matrix (DSM) that is a different representation for the development view. Each line and column represents a package. For instance, row 1 and column 1 represent the package editor. This package in the first row calls classes in the packages entity (column 02) and game (column 03). The relationship between these packages is variable because they are implemented only by some of the variants. We used the letter V to indicate the variability, which is equivalent to the red-dashed lines in Fig. 5. On the other hand, the relationship from game (row 3) to package entity (column 02) is mandatory because it is implemented by all the variants. We used the letter M to represent the mandatory relationships, which is equivalent to the blue solid lines in Fig. 5.

We highlight that the implementation of mandatory packages such as the entity, game, and main can present variation points. In other words, a mandatory package can implement variable classes. The same happens when we analyze mandatory classes. Such classes can present variation points. However, in the case of the classes, the variability happens in lower level (such as source code statements inside a method). Our objective is to identify the variability in a higher level of abstraction. In this context, the low-level variability did not affect the structure of the PLA.

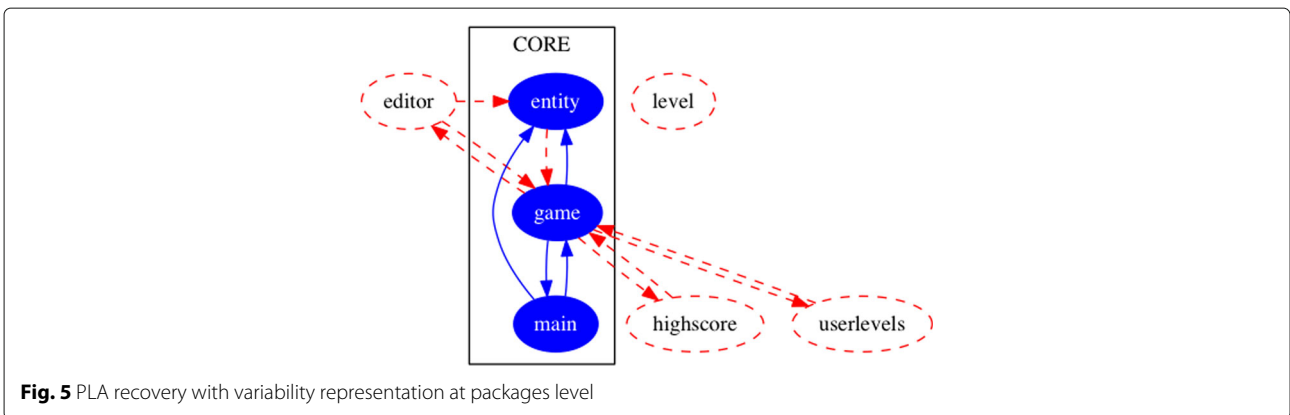
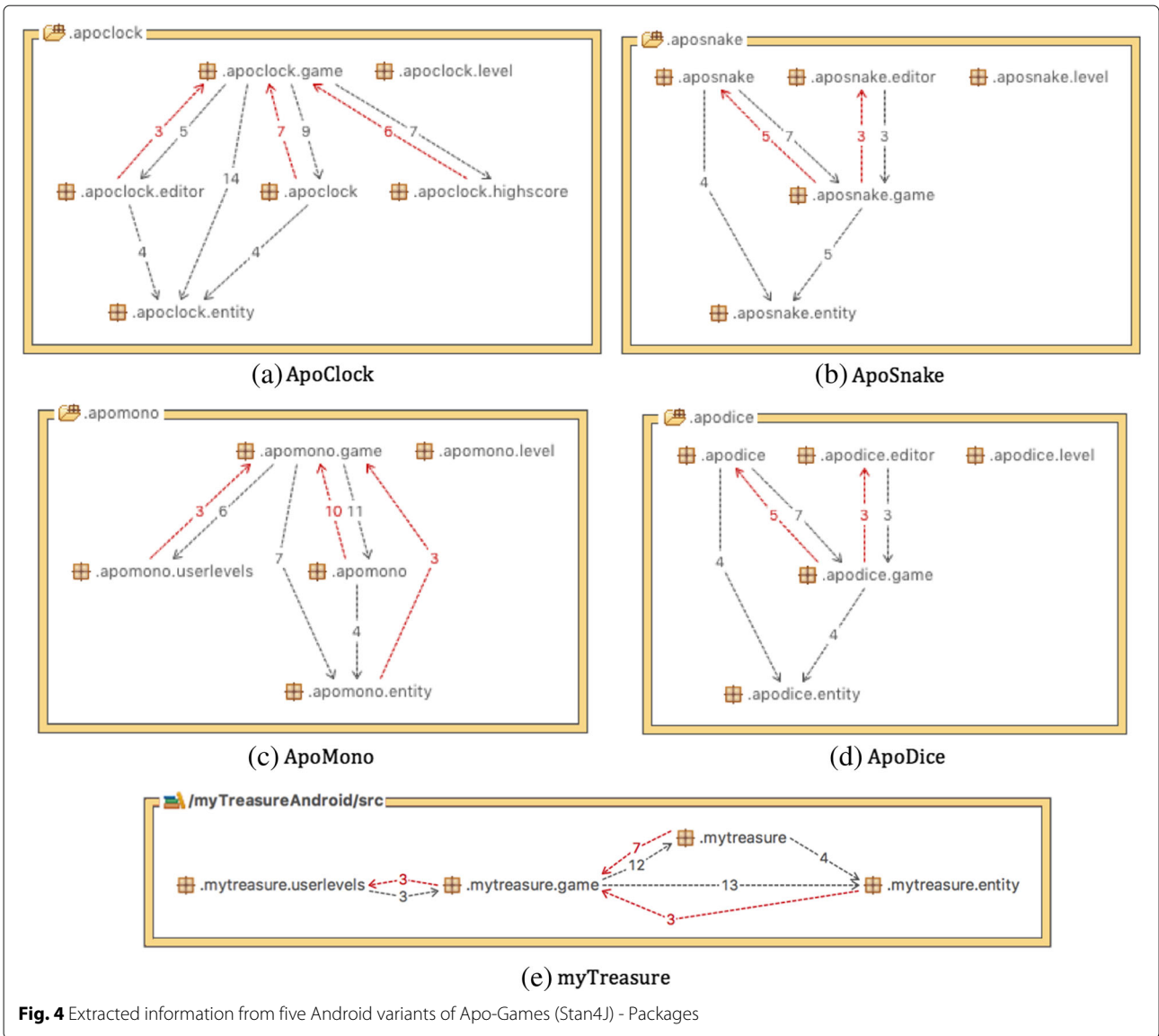
**Study design**

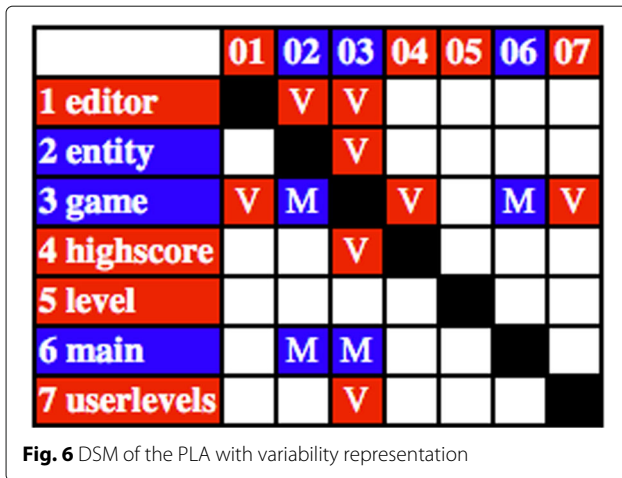
This section describes the evaluation design based on the goal/question/metric (GQM) approach [32].

**Evaluation goal:** *Evaluating how the proposed approach supports the cost-effective product line architecture recovery by the identification and filtering of outliers.*

**Research questions (RQ):** Guided by our evaluation goal, we derived the following research questions.

- *RQ1: How similar and how different are the variants?* Since our goal is to deal with outliers, first we need to figure out the degree of similarity and variability of the variants to determine which variants are, potentially, too expensive to be included in the PLA recovering process.





- *Metric:* We relied on the analysis of the Jaccard similarity <sup>5</sup>. The Jaccard similarity measure is defined as the size of the intersection divided by the size of the union of the sample sets. In our case, the sample sets will be the PLA components of each pair of variants.
- *RQ2: Is there any correlation between the size of the game variants and the existence of outliers?* We aim to investigate which are the characteristics that can help to identify outlier variants. For instance, are variant outliers either the bigger or smaller games? To what extent the variability of a game makes it too different from other variants?
  - *Metric:* We applied the correlation analysis between the variants size (lines of code) and the number of packages and classes exclusive for a variant. For this analysis, we used the Pearson correlation coefficient [33].
- *RQ3: To what extent eliminating outliers support recovering better PLAs?* We aim to analyze the quality of the obtained solutions when removing outliers. The quality of such solutions is evaluated according to four architectural metrics. Regarding the implementation level, we want to investigate the impact of outliers removal in class level and in package level.
  - *Metric:* We considered the SSC, SVC, RSC, and RVC metrics, described in the “Background” section. We collected these metrics as a result of our approach. We applied the threshold analysis and collected the metrics after the threshold cut.

Table 2 presents a summary of the GQM method for our evaluation

**Table 2** Describing the study according to GQM

Goal	Evaluating cost-effectiveness of the proposed approach
Purpose	Analyze the impact of outliers filtering
With respect to	PLA recovering
From the point of view	Architects, engineers, and programmers
In the context of	SPL extraction from Apo-Games variants
Question	Metric
RQ1	Jaccard similarity
RQ2	LOC, number of packages and classes exclusive to a variant
RQ3	SSC, SVC, RSC, RVC

### Case study

To answer the research questions, we considered the Apo-Games variants as an input for our approach. Initially, we composed a set of 34 variants, where 25 were from a repository and nine from the Apo-Game webpage (see “Background” section). Next, we describe how we selected the variants for the both package analysis and class analysis.

For the package analysis, we excluded two variants because they did not provide information for allowing the information extraction ②. Concretely, we identified that the developer did not use the package structure in the implementation of the projects ApoCheating and Tutorvolley. Moreover, for the package analysis, we excluded ApoDefence variant because most of the source code was obfuscated by the developer and we did not have access to the original source code for this variant; however, this variant is considered in the class analysis.

For the class analysis, we selected 13 variants that were available as Eclipse projects and four variants with Java class files. The Jar files were not included in this analysis, because the extraction tool did not support automatic extraction of classes information based on Jar files.

Table 3 presents the information of the selected variants. We managed to extract 31 TGF files with package information and 17 TGF files with class information.

### Threshold configurations

In the context of this work, a threshold is used to indicate which elements, namely classes and packages, will not be used as input to recover PLAs. In other words, the threshold allows us to filter exclusive packages and classes without eliminating the outlier variants.

To determine the threshold configurations to be taken into account during the evaluation, we rely on a report generated by our recovery approach. This report presents the frequency that each class and/or package exists in



**Table 3** Apo-Games projects—metrics' summary

Projects	Plat.	Year	LOC	#E	#Jr	#Ja	#T	#TJ	#P	#EP	%P	#C	#EC	%C
ApoDefense	Desktop	2007	12917	-	✓	✓	-	✓	-	-	-	66	56	85%
ApoMushroom	Desktop	2007	7789	-	✓	-	✓	-	6	1	16%	-	-	-
ApoSkunkman	Desktop	2007	8645	-	✓	-	✓	-	16	6	37%	-	-	-
ApoSheep	Desktop	2007	5129	-	✓	-	✓	-	8	0	-	-	-	-
ApoPrism	Desktop	2008	8948	-	✓	-	✓	-	11	1	9%	-	-	-
ApoStarz	Desktop	2008	6454	-	-	✓	✓	✓	11	1	9%	49	8	16%
ApoBot	Desktop	2009	5857	-	-	✓	✓	✓	8	0	-	48	2	4%
ApoDoor	Desktop	2009	7052	-	✓	-	✓	-	7	0	-	-	-	-
ApoPolarium	Desktop	2009	8825	-	✓	-	✓	-	9	0	-	-	-	-
ApoSliding	Desktop	2009	12032	-	✓	-	✓	-	8	0	-	-	-	-
ApoSoccer	Desktop	2009	10736	-	✓	-	✓	-	18	10	55%	-	-	-
ApoWomanInv	Desktop	2009	9696	-	✓	-	✓	-	9	0	-	-	-	-
ApoCommando	Desktop	2010	9820	✓	-	-	✓	✓	5	0	-	72	18	25%
ApolceJumpR.	Desktop	2010	8138	-	✓	-	✓	-	9	0	-	-	-	-
ApoPongBeat	Desktop	2010	6591	✓	-	-	✓	✓	10	1	10%	79	31	39%
Apolcarus	Desktop	2011	5851	✓	-	-	✓	✓	9	0	-	59	15	25%
ApoMarc	Desktop	2011	5493	-	-	✓	✓	✓	10	2	20%	59	6	10%
ApoMario	Desktop	2011	17184	-	✓	-	✓	-	14	2	14%	-	-	-
ApoSlitherLink	Desktop	2011	7313	✓	-	-	✓	✓	8	0	-	62	8	12%
ApoNotSoSimple	Desktop	2011	7558	✓	-	-	✓	✓	10	0	-	57	1	2%
ApoRelax	Desktop	2011	6688	✓	-	-	✓	✓	10	0	-	56	3	5%
ApoSimple	Desktop	2011	19558	✓	-	-	✓	✓	16	6	37%	104	48	46%
ApoSnake	Desktop	2012	6557	-	✓	-	✓	-	10	0	-	-	-	-
ApoSudoku	Desktop	2012	5517	✓	-	-	✓	✓	9	0	-	41	2	5%
Apolmp	Desktop	2012	6432	-	✓	-	✓	-	12	1	8%	-	-	-
ApoClock	Mobile	2012	3615	✓	-	-	✓	✓	6	1	16%	28	0	-
ApoDice	Mobile	2012	2523	✓	-	-	✓	✓	5	0	-	19	0	-
ApoSheeptastic	Desktop	2012	46704	-	✓	-	✓	-	18	6	33%	-	-	-
ApoSnake	Mobile	2012	2965	✓	-	-	✓	✓	5	0	-	19	0	-
ApoMono	Mobile	2013	6487	✓	-	-	✓	✓	5	0	-	26	0	-
myTreasure	Mobile	2013	5360	✓	-	-	✓	✓	4	0	-	27	0	-
ApoTreasure	Desktop	2014	8205	-	✓	-	✓	-	10	0	-	-	-	-
Total	-	-	178259	13	16	4	31	17	296	38	13%	871	198	26%

LOC lines of code, Plat. platform, #E projects available in Eclipse, #Jr projects available in Jar files, #Ja projects available in Java files, #T TGF file packages, #TJ TGF file classes, #P number of packages, #EP number of exclusive packages, %P percentage of exclusive packages over the total, #C number of classes, #EC number of exclusive classes, %C percentage of exclusive classes over the total

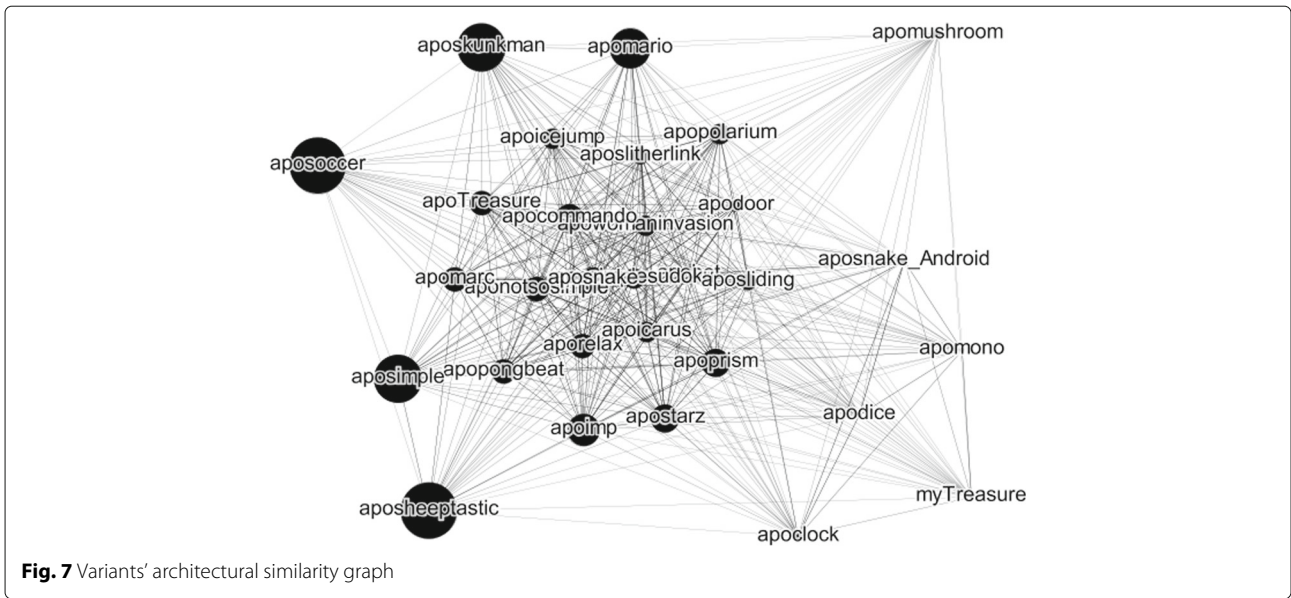
the variants. For example, assuming we are dealing with 17 variants, when a class/package X exists in the implementation of all variants, its percentage of appearance frequency is 100%. On the other hand, if a class/package Y is in only one variant, then its appearance frequency is only 6%.

Based on the report of appearance frequency, we determine the threshold configurations. Recalling the example presented in the last paragraph, if we have a threshold

configuration of 6% for the recovering process, means that class existent in only one variant will not be considered for constructing the PLA.

### Results and analysis

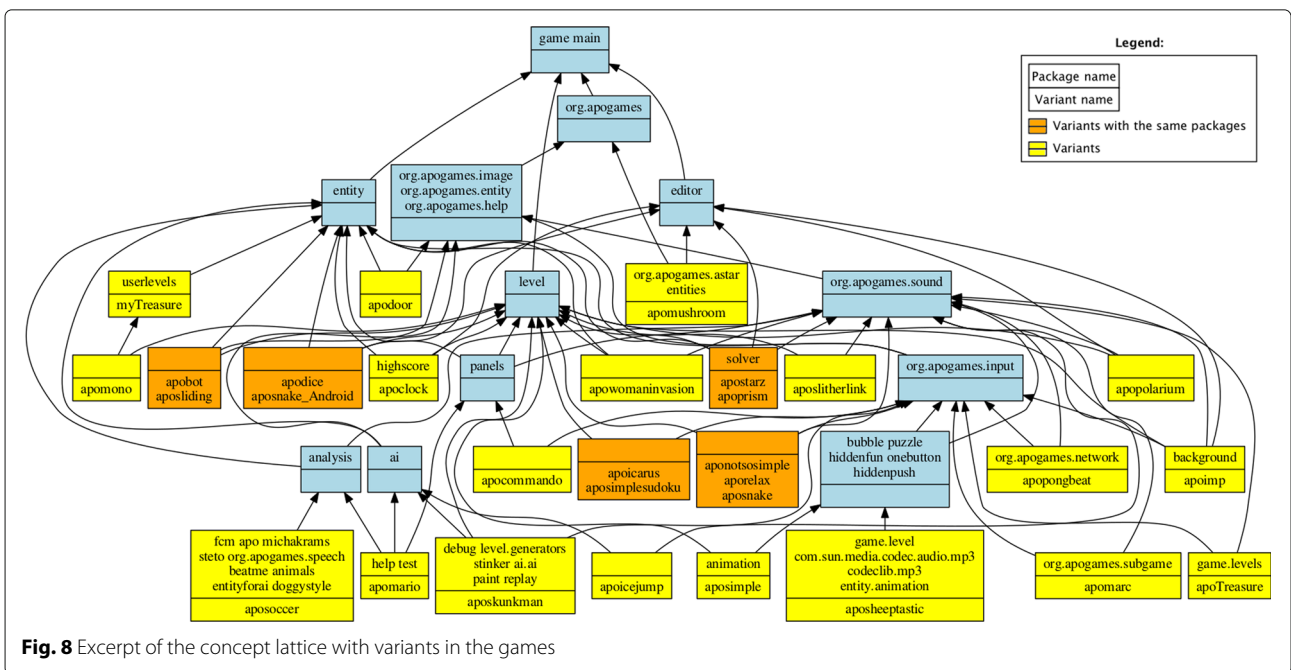
In this section, we describe the study results. We performed an initial analysis of the variants. Figure 7 shows a graph where the nodes are the variants and the size of the nodes are related to the number of packages of



each variant. Edges exist between nodes when the Jaccard similarity between them is higher than zero. This similarity determines the weight of the edges which is used by the automatic layout of the graph to approximate similar variants and to keep off the variants which are different. On the left side, we observe five variants of large size (ApoMario, ApoSoccer, ApoSimple, ApoSkunkman, and ApoSheeptastic) which are dissimilar among them and among the rest of the variants. We also observe, on the right side, the same case with six small variants (ApoMushroom, ApoSnake\_Android, ApoMono, ApoDice, ApoClock, and myTreasure).

On the contrary, around the center of the figure, we can observe some variants which are quite similar with large similarity weight represented with the size of the edges.

We performed FCA to automatically obtain the representation shown in Fig. 8, which is known as the pruned concept hierarchy [34]. It allows us to identify that some variants are overlapped, which shows that there are certain games with almost the same structure. The ApoBot and ApoSliding variants on the middle of the figure illustrates such a case. By recursively following the arrows

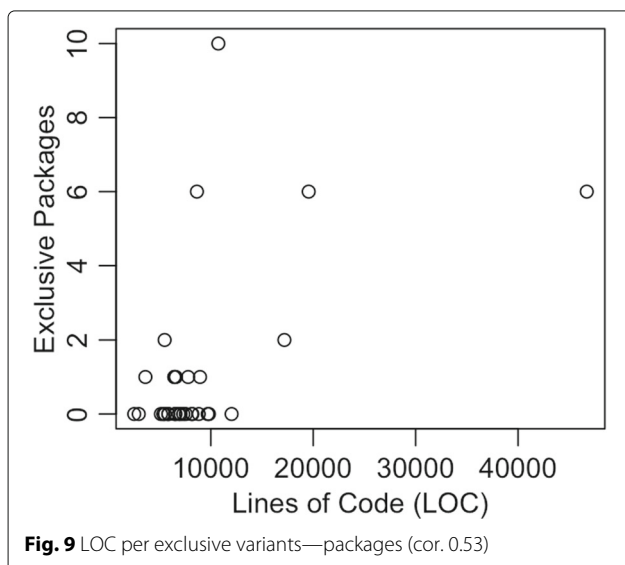


until the root, we can know that these variants consists of the packages `level`, `entity`, and a set of common packages for all variants (`game`, `org.apogames.image` etc.).

In addition, we can find the variants `ApoIcarus` and `ApoSudoku` that are grouped in the same concept as they consist of the same packages, and we can observe how some variants have packages that are exclusively specific to one variant (e.g., `ApoMario` has the packages `help` and `test` which do not appear in any other variant). Both Figs. 7 and 8 are helpful to understand how similar are Apo-Games variants among them, to visually identify outliers and clusters, and to understand how packages are distributed among the variants. It is out of the scope of the paper to evaluate these already existing visualization paradigms. We presented them as they are helper artifacts for the manual analysis during the execution of our approach.

Figures 9 and 10 show the correlation between lines of code (LOC) and exclusive variants' information (packages and classes implemented for a specific variant). Each point corresponds to the variants. For instance, we can observe the variant `ApoSheeptastic` (shown in Fig. 9 with six exclusive packages on y axis and 46704 LOC on x axis). In addition, in Fig. 10, we can observe the variant `ApoSimple` with 48 exclusive classes on y axis and 19558 LOC on x axis.

Figure 9 shows the correlation between exclusive packages and the size of the variants, and Fig. 10 shows the correlation between exclusive classes and the variants' size. For the analysis of the correlation, we used Pearson coefficient [33]. The values for this coefficient varies from -1 (weak correlation) to 1 (strong correlation). The correlation among exclusive packages and variants LOC is 0.53,



meaning that there is a strong correlation. Moreover, the correlation among exclusive classes and variants' size is even stronger, being equal to 0.80.

Table 4 presents the collected metrics for threshold analysis of packages. We executed the PLA recovery approach five times according to threshold values based on a report generated by the PLA recovery approach. The report identifies the packages and classes according to their existence in the variants. For instance, when the package or class is implemented in all the products, the report informs that this package or class appears in the implementation of 100% of the variants.

When we eliminated packages exclusive to only one variant (threshold of 4%), the number of optional packages dropped from 48 to 19 packages. In other words, it reduced the variability (“noise”) in the PLA representation. Moreover, it improved the SSC and SVC metrics' values. We performed this activity manually in our previous study [28] and it was time-consuming. By applying the threshold technique, the time and effort were reduced.

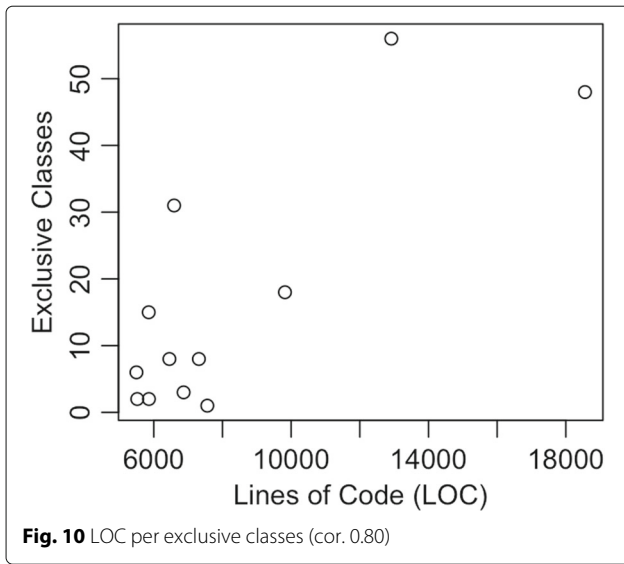
Table 5 presents the collected metrics for threshold analysis of classes. We executed the PLA recovery approach eleven times according to different threshold values. When we reduced the abstraction level to classes, we identified a higher granularity in variability (278 optional classes and only 2 mandatory classes).

As opposed to the analysis of packages, the threshold technique was not so efficient to reduce the amount of variability in the PLA representation. The metrics' values did not change over the PLA recovery using different threshold values. We believe that this happened because of the lower number of mandatory classes.

In the report, we identified some classes that are present in 91% of the variants. For this reason, we used a different strategy to improve the PLA representation. We eliminated the variants with a large number of exclusive classes to raise the number of mandatory classes and improve SSC and SVC metrics' values.

Table 6 presents the combination of eliminating variants and then applying the threshold. We identified that by eliminating one variant, the number of mandatory classes raised from 2 to 19. However, even eliminating 4 variants with the largest number of exclusive classes, the SSC and SVC metrics did not change. We identified metric improvements when we applied the threshold.

Another analysis we could perform with the inclusion of new Apo-Games variants is the identification of variability and commonalities between desktop and mobile implementations of a same game. Figure 11 presents the recovered PLA for `ApoSnake` using the mobile and desktop variants. Figure 11a shows the core elements implemented by both variants (`main`, `entity`, `game`, and `level`), b shows the exclusive packages implemented by the mobile variant (`editor`), and (c) shows the exclusive packages



implemented by the desktop variant (packages from the `org.apogames` library). We also identified the same pattern between desktop and mobile implementation in MyTreasure and ApoTreasure variants.

**Discussion**

In this section, we interpret the results and discuss the findings by answering the research questions.

**Answering RQ1**

Regarding the analysis of how similar and how different are the variants, the Jaccard similarity measure indicated that five variants can be considered as outliers, namely ApoSoccer, ApoSimple, ApoSkunkman, ApoSheeptastic, and ApoMario, because they are dissimilar among the other variants. In other words, it will be too costly to be included in the PLA recovery because they include a high number of exclusive packages (ApoSoccer = 55%, ApoSimple = 37%, ApoSkunkman = 37%, and ApoSheeptastic = 33%). Unfortunately, it was not possible to investigate the classes from ApoSoccer, ApoSkunkman, ApoSheeptastic, and ApoMario because the Java files were

**Table 4** Recovered metrics from the PLAs (packages)

TH	SSC	SVC	RSC	RVC	CO	OR	CM	R
00%	0.04	0.96	0.01	0.99	48	193	2	1
04%	0.1	0.9	0.02	0.98	19	84	2	1
07%	0.2	0.8	0.03	0.97	10	46	2	1
10%	0.2	0.8	0.03	0.97	9	39	2	1
26%	0.2	0.8	0.03	0.97	8	23	2	1

TH threshold, SSC structure similarity coefficient, SVC structure variability coefficient, RSC relation similarity coefficient, RVC relation variability coefficient, CO PackageOptional, OR OptionalRelation, CM PackageMandatory, R MandatoryRelation

**Table 5** Recovered metrics from the PLAs (classes)

TH	SSC	SVC	RSC	RVC	CO	OR	CM	R
00%	0.01	0.99	0.00	1.00	278	625	2	0
09%	0.02	0.98	0.00	1.00	80	114	2	0
17%	0.03	0.97	0.00	1.00	68	80	2	0
26%	0.04	0.96	0.00	1.00	54	58	2	0
34%	0.04	0.96	0.00	1.00	51	47	2	0
42%	0.05	0.95	0.00	1.00	46	37	2	0
59%	0.05	0.95	0.00	1.00	37	23	2	0
67%	0.07	0.93	0.00	1.00	27	9	2	0
76%	0.08	0.92	0.00	1.00	24	8	2	0
89%	0.10	0.90	0.00	1.00	18	7	2	0
92%	1.00	0.00	n.a.	n.a.	0	0	2	0

Legends: TH threshold, SSC structure similarity coefficient, SVC structure variability coefficient, RSC relation similarity coefficient, RVC relation variability coefficient, CO ClassOptional, OR OptionalRelation, CM ClassMandatory, R MandatoryRelation

not available. However, we believe they followed the same pattern as ApoSimple with a high number of exclusive classes (46% of the classes).

Moreover, we identified that the mobile variants are dissimilar (see Fig. 7 on the right side) because these variants lack the implementation of the package `org.apogames`. We believe this happened because of the limitations imposed by the Android framework.

**Answering RQ2**

Concerning the correlation between the size of the game variants and the existence of outliers, the data reveal that for the Apo-Games case study, the differentiation among variant implementation increases with their size. This is confirmed by the correlation between the metrics' LOC and exclusive variants, which correlate highly. We can explain this correlation with the observation that larger games usually implement more complex mechanisms and, consequently, are more variable. On the other hand, smaller games tend to share the same structure (architecture).

Figure 12 shows the DSMs of the Android (i.e., mobile) and Java (i.e., desktop) projects side by side. We identified that five packages (`editor`, `entity`, `game`, `level`, and `main`) were implemented in both mobile and desktop games. As differences between them, we highlight the higher number of function calls in desktop games and mobile games has three variants (`editor`, `highscore`, and `userlevels`). Moreover, desktop projects presented six packages from `org.apogames`. If we eliminate these packages from the comparison, both sets presented similar PLA.

This might have happened because the developer used the Java project as a basis for the creation of the Android projects. Moreover, the limitations imposed by

**Table 6** PLA metrics after eliminating some variants and threshold analysis (classes)

EV	SSC	SVC	RSC	RVC	CO	OR	CM	R
0	0.01	0.99	0.00	1.00	278	625	19	7
1	0.08	0.92	0.02	0.98	205	465	19	7
2	0.10	0.90	0.02	0.98	157	365	19	7
3	0.13	0.87	0.02	0.98	126	313	19	7
4	0.15	0.85	0.02	0.98	107	263	19	7
Application of the threshold analysis								
13%	0.26	0.74	0.07	0.93	56	91	19	7
26%	0.32	0.68	0.20	0.80	42	56	19	7
38%	0.38	0.62	0.20	0.80	32	37	19	7
51%	0.46	0.54	0.30	0.70	23	18	19	7

Legends: EV eliminated variants, SSC structure similarity coefficient, SVC structure variability coefficient, RSC relation similarity coefficient, RVC relation variability coefficient, CO ClassOptional, OR OptionalRelation, CM ClassMandatory, R MandatoryRelation

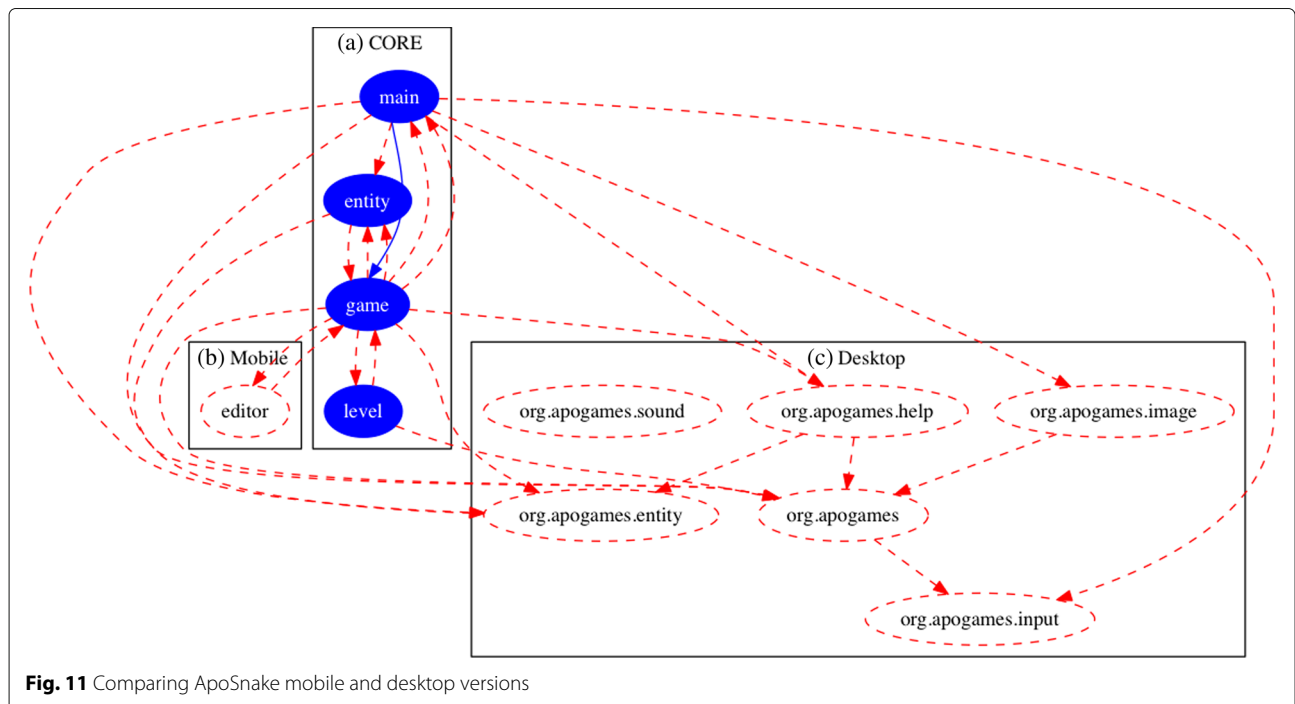
the Android framework could lead to the simplification of the projects. For this reason, we did not identify packages from `org.apogames` in the Android projects as can be seen in Fig. 11. Moreover, Fig. 12 highlights the evidence that bigger projects implement more complex logic which demands the implementation of exclusive elements.

**Answering RQ3**

To figure out the impact of outlier elimination to support the recovery of better PLAs, we analyzed the threshold technique results. The implementation of this technique allowed the reduction of the number of exclusive packages and classes without eliminating the outlier variants. In the

high-level context (packages), we identified the reduction of the exclusive packages and the balance between SSC and SVC metrics in the first cuts of the threshold. However, in the lower level (classes), the number of exclusive classes raised due to the granularity of the implementation. We believe that behavior happened because it is easier to maintain the organization in the packages than in the classes. For this reason, we eliminated four outlier variants with a high number of exclusive classes. Only after eliminating these outliers, we identified the improvements in the metrics and in the PLA representation.

Figure 13 presents the application of three threshold values (0%, 4%, and 7%). The first execution of the threshold



**Fig. 11** Comparing ApoSnake mobile and desktop versions



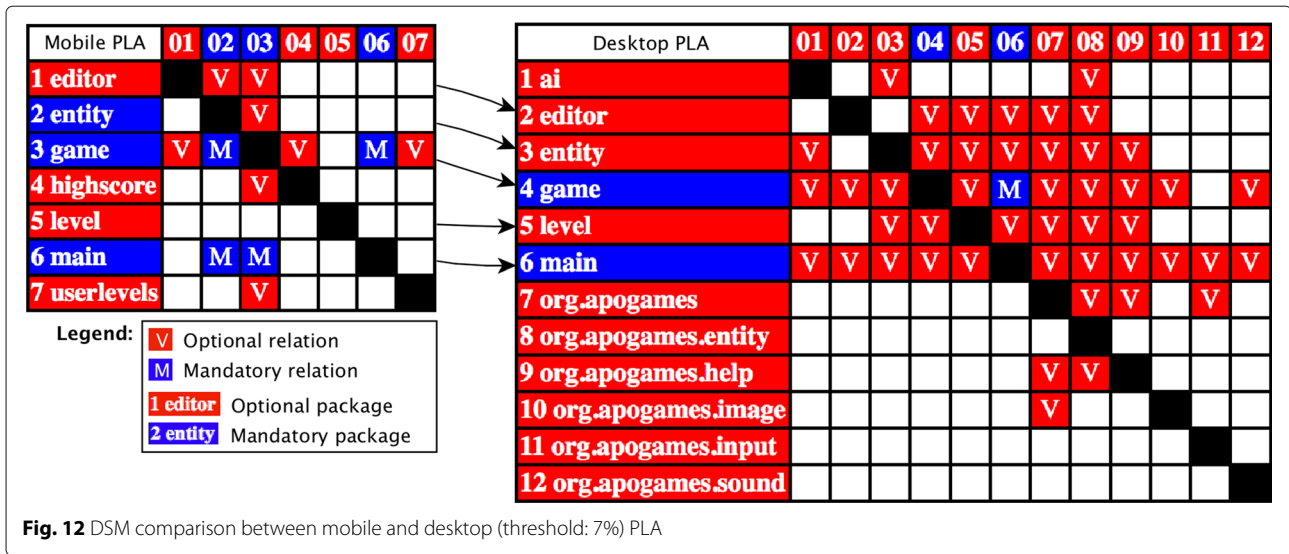


Fig. 12 DSM comparison between mobile and desktop (threshold: 7%) PLA

technique reduced the DSM’s size in 58% (see Fig. 13b–threshold 4%). Moreover, the technique second execution reduced the DSM’s size in 76% (see Fig. 13c–threshold 7%).

These results indicates the efficiency of the threshold technique in filtering outliers. It maintains the mandatory packages, preserves the packages implemented in the majority of the variants, and eliminates the packages exclusive to a small number of variants.

**Threats to validity**

The following threats to validity are discussed to reveal their potential interference with our study design.

**Internal validity**

*Selection.* Depending on how the subjects are selected from a larger group, the selection effects can vary. We identified this effect during the selection of the variants in the class analysis. When we considered all the variants in the recovery, the PLA was composed by only optional classes. To reduce the noise in the representation, we eliminated the variants with a high number of exclusive classes. Moreover, due to extraction tool limitation and unavailability of the source code of some projects, we cannot extract the class information of all the variants. To minimize the threats, we improved the approach for this study based on the evidence we gathered by applying the approach in other settings. We considered SPL projects from different domains and implemented with different variability mechanisms [13, 27, 28].

**External validity**

*Interaction of selection and treatment.* This is an effect of having a subject population not representative of the population we want to generalize. The Apo-Games represents a small portion of the domain we want to

generalize (similar variants that can be used to migrate to SPL domain). However, it is a case that can help in the evidence building regarding the impact of the variability in the context of PLA recovery. To minimize the threats associated with the use of the approach in other settings, we reviewed the evidence raised when we performed an exploratory study to investigate the PLA recovery using SPL products as variants [27]. We validated the guideline to support our approach. Moreover, we found an issue related to packages and classes implementing the same logic but using different names. We eliminated information specific to projects to reduce the impact of this issue and we intend to automate this activity in future work.

**Construct validity**

*Mono-operation bias.* We only considered subjects of the Apo-Games in the case study. It may under-represent the construct and thus not give the full picture of the problem. The projects evolved over the years, and new ideas were included contributing for the maturity and raise of the complexity of the projects. To mitigate this threat, we included bigger variants and variants from mobile domain.

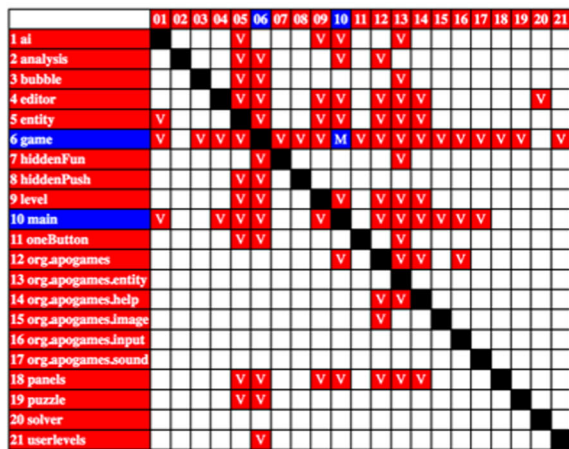
*Inadequate preoperational explication of constructs.* The constructs are not sufficiently defined before they are translated into measures or treatments. The theory is not clear enough regarding the automation effectiveness of PLA recovery and improvement of the recovered PLA. We based our analysis on metric analysis and PLA representation. Even though, it is still impossible to eliminate human intervention.

**Conclusion validity**

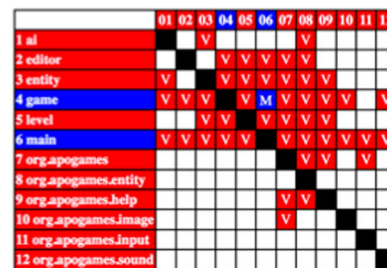
*Low statistical power.* One of the threats of our previous study [13] was the sample size (20 variants). To mitigate this issue, we included 14 new variants in this study. From



(a) Threshold: 0%



(b) Threshold: 4%



(c) Threshold: 7%

Fig. 13 DSMs showing the application of the threshold

the 34 variants, we considered 31 in the package analysis and 17 in the classes analysis. However, as the purpose was to improve the initial evidence on the existence of a correlation between variant size and exclusive components, and investigate how the threshold improves the effectiveness of the PLA recovery, we still understand that for generalizing such findings, we need a sample with new projects.

### Related work

As mentioned previously, PLA recovery techniques have been overlooked and the analysis of commonality and variability has been more focused on low-level implementation elements [8]. However, in this section, we present a related work on PLA recovery.

Shatnawi et al. [35] proposed an approach to PLA recovery that focused on the comparison of components (classes and interfaces) recovered from different versions of the same SPL. The authors relied on FCA to identify the variability and created a variability model. Our study did not use different variants derived from an already existing SPL. Instead, several variants from the same domain were used as an input to our approach. For each variant, we extracted structural information from source code and collected information about variability found within classes, packages, and their relationships. Finally, they did not consider the removal of outliers through thresholds.

Linsbauer et al. [36] presented an approach for extracting information from sets of related product variants—structural information from SPL products source code—with the goal of recovering a feature model for the SPL. Likewise, our approach supports the extraction of structural information from the source code of sets of related variants, but with the goal of recovering a PLA for the SPL.

Fenske et al. used the Apo-Games case study to evaluate variant-preserving refactorings for SPL adoption [22]. In our approach, we performed renaming refactorings (manually) while eliminating information specific to variants. However, our approach is more focused on effective extraction of PLAs which can consist on the exclusion of variants, packages, or classes from the analysis.

In our previous work [13], we propose an automatic PLA recovery approach for the identification of the minimum subset of cross-product architectural information. We evaluated the approach by considering 20 open-source variants of the Apo-Games project. In this paper, we extend our work by including further details of the proposed approach. We introduced a guideline to support the PLA recovering process, we included 14 Apo-Games new variants (9 desktops and 5 mobiles) in addition to the 20 ones of our previous study, and we performed an in-depth analysis of the results.

### Concluding remarks

PLA recovery can provide useful information for defining the foundations to facilitate SPL adoption. Instead of

working from scratch, the recovered PLA can provide initial support to development and maintenance tasks. In this context, one of the main issues is to manage the variability introduced by some variants (i.e., outliers).

In this paper, we propose an approach for recovering the PLA based on the source code of similar variants. We performed a FCA to identify the outliers. Moreover, we implemented the threshold analysis to reduce the number of exclusive components without eliminating the variants of the recovered PLA.

We evaluated the approach using the Apo-Games projects. We identified that five variants can be considered as outliers because they were dissimilar among the other variants. Moreover, the data reveal that the differentiation among variant implementation increases with their size. In other words, bigger projects lead to the implementation of exclusive components. We reduced the number of exclusive components by applying the threshold technique, and the results indicated the efficiency of the technique in filtering outliers.

As future work, we intend to apply our approach in other case studies to extend the sample size and strength of the analysis of the evidence. We aim to improve our approach with search-based software engineering techniques to automate the definition of the threshold values. Moreover, we intend to improve the PLAR tool for automatically eliminate specific information of the variants.

### Endnotes

- <sup>1</sup> <http://apo-games.de>
- <sup>2</sup> [https://bitbucket.org/Jacob\\_Krueger/apogamessrc](https://bitbucket.org/Jacob_Krueger/apogamessrc)
- <sup>3</sup> <http://stan4j.com/>
- <sup>4</sup> <https://but4reuse.github.io/>
- <sup>5</sup> [https://en.wikipedia.org/wiki/Jaccard\\_index](https://en.wikipedia.org/wiki/Jaccard_index)

### Acknowledgements

We would like to acknowledge the institutions which contributed by providing support to the execution of this work, namely INES 2.0, CNPq, and FAPESB.

### Authors' contributions

CL, WA, and JM designed the outlier filtering approach and reported it in this manuscript. CL conducted the PLA recovery experimentation and obtained the data for figure and table creation. All authors reviewed the final manuscript and discussed the results.

### Author's information

Ph.D. Crescencio Lima is a professor at the Federal Institute of Bahia (IFBA), Ph.D. Wesley KG Assunção is a professor at Federal University of Technology - Paraná (UTFPR), Ph.D. Jabier Martinez is a researcher at Tecnalia, M.D. William Mendonça is lecturer at Federal University of Technology - Paraná (UTFPR), Ph.D. Ivan Machado and Ph.D. Christina Chavez are professors at the Federal University of Bahia (UFBA).

### Funding

This research was partially funded by INES 2.0; CNPq grants 465614/2014-0 and 408356/2018-9; and FAPESB grants JCB0060/2016 and BOL2443/2016.

### Availability of data and materials

Not applicable.

**Competing interests**

The authors declare that they have no competing interests.

**Author details**

<sup>1</sup>Federal University of Bahia, Ademar de Barros - Campus de Ondina, Salvador, 40170-110, Brazil. <sup>2</sup>Federal Institute of Bahia, Sérgio Vieira de Mello Avenue, 3150 - Zabelê, Vitória da Conquista, 45078-900 Brazil. <sup>3</sup>Federal University of Technology - Paraná, Cristo Rei Str, 19 - Vila Becker, Toledo, 85902-490 Brazil. <sup>4</sup>Tecnalia, Arteaga Auzoa, 25, 700, E-48160 Derio, Spain.

Received: 6 February 2019 Accepted: 29 May 2019

Published online: 24 June 2019

**References**

- van der Linden FJ, Schmid K, Rommes E (2007) Software product lines in action: the best industrial practice in product line engineering. Springer
- Pohl K, Böckle G, van der Linden F (2005) Software product line engineering: foundations, principles, and techniques. Springer-Verlag New York Inc
- Apel S, Batory D, Kastner C, Saake G (2013) Feature-Oriented Software Product Lines
- Holmes R, Walker RJ (2013) Systematizing pragmatic software reuse. *ACM Trans Softw Engineer Methodol* 21(4):20:1–20:44
- Dubinsky Y, Rubin J, Berger T, Duszynski S, Becker M, Czarnecki K (2013) An exploratory study of cloning in industrial software product lines. *IEEE Computer Society, Washington, DC*
- Kulkarni N, Varma V (2017) Perils of opportunistically reusing software module. *Softw: Pract Exp* 47(7):971–984
- Assunção WKG, Lopez-Herrejon RE, Linsbauer L, Vergilio SR, Egyed A (2017) Reengineering legacy applications into software product lines: a systematic mapping. *Empirical Softw Engineer* 22(6):2972–3016
- Martinez J, Assunção WKG, Ziadi T (2017) ESPLA: a catalog of extractive SPL adoption case studies. In: *Proceedings of the 21st International Systems and Software Product Line Conference, SPLC 2017*. ACM Vol. B, pp 38–41
- Berger T, Rublack R, Nair D, Atlee JM, Becker M, Czarnecki K, et al (2013) A survey of variability modeling in industrial practice. In: *Proceeding of the 7th International Workshop on Variability Modelling of Software-intensive Systems*
- Wille D, Önder B, Cleophas L, Seidl C, van den Brand M, Schaefer I (2018) Improving custom-tailored variability mining using outlier and cluster detection. *Sci Comput Program* 163:62–84
- Taylor RN, Medvidovic N, Dashofy EM (2010) *Software architecture-foundations, theory, and practice*. Wiley
- Garcia J, Ivkovic I, Medvidovic N (2013) A comparative analysis of software architecture recovery techniques. In: *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013*. IEEE, Silicon Valley, pp 486–496
- Lima C, Assunção WKG, Martinez J, do Carmo Machado I, von Flach G, Chavez C, Mendonça WDF (2018) Towards an automated product line architecture recovery: the Apo-Games case study. In: *VII Brazilian Symposium on Software Components, Architectures, and Reuse. SBCARS '18*. ACM, pp 33–42
- Krüger J, Fenske W, Thüm T, Aporius D, Saake G, Leich T (2018) Apo-Games - a case study for reverse engineering variability from cloned Java variants. In: *Proceedings of the 22nd International Systems and Software Product Line Conference - Challenge Track. SPLC '18*. ACM. Available from: <https://variability-challenges.github.io/2018/ApoGames>
- Galster M, Weyns D, Avgeriou P, Becker M (2013) Variability in software architecture: views and beyond. *SIGSOFT Softw Eng Notes* 37(6):1–9
- Chen L, Ali Babar M (2011) A systematic review of evaluation of variability management approaches in software product lines. *Inf Softw Technol* 53(4):344–362
- Ahmed F, Capretz LF (2008) The software product line architecture: an empirical investigation of key process activities. *Inf Softw Technol* 50(11):1098–1113
- Verlage M, Kiesgen T (2005) Five years of product line engineering in a small company. In: *Proceedings of the 27th International Conference on Software Engineering*. ACM, pp 534–543
- Kruchten P (1995) The 4+1 View Model of Architecture. *IEEE Softw* 12(6):42–50
- Ducasse S, Pollet D (2009) Software architecture reconstruction: a process-oriented taxonomy. *IEEE Trans Softw Engineer* 35(4):573–591
- Zhang T, Deng L, Wu J, Zhou Q, Ma C (2008) Some metrics for accessing quality of product line architecture. In: *International Conference on Computer Science and Software Engineering*. IEEE Vol. 2, pp 500–503
- Fenske W, Meinicke J, Schulze S, Schulze S, Saake G (2017) Variant-preserving refactorings for migrating cloned products to a product line. In: *IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017*. IEEE Computer Society, Klagenfurt, pp 316–326
- Roughan M, Tuke SJ (2015) Unravelling Graph-Exchange File Formats. [abs/1503.02781](https://arxiv.org/abs/1503.02781)
- Cardoso MPS, Lima C, Chavez C, do Carmo Machado I (2017) PLAR tool – a software product line architecture recovery tool. In: *8th Brazilian Conference on Software: Theory and Practice - Tool Session*
- Martinez J, Ziadi T, Bissyandé TF, Klein J, Traon YL (2017) Bottom-up technologies for reuse: automated extractive adoption of software product lines. In: *Proceedings of the 39th International Conference on Software Engineering, ICSE 2017*. Companion Volume IEEE Computer Society, Buenos Aires, pp 67–70
- Lima-Neto CR, Cardoso M, Chavez C, Almeida E (2015) Initial evidence for understanding the relationship between product line architecture and software architecture recovery, Vol. IX. *Brazilian Symposium on Components, Architectures and Reuse Software (SBCARS)*
- Cardoso MPS, Lima C, de Almeida ES, do Carmo Machado I, von Flach G, Chavez C (2017) Investigating the variability impact on the recovery of software product line architectures: an exploratory study. In: *Proceedings of the 11th Brazilian Symposium on Software Components, Architectures, and Reuse*. ACM, pp 12:1–12:10
- Lima C, do Carmo Machado I, de Almeida ES, von Flach Garcia Chavez C (2018) Recovering the product line architecture of the Apo-Games. In: *Proceedings of the 22nd International Systems and Software Product Line Conference SPLC '18*. ACM
- Rubin J, Chechik M (2012) Locating distinguishing features using diff sets. In: *27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, pp 242–245
- Fischer S, Linsbauer L, Lopez-Herrejon RE, Egyed A (2014) Enhancing clone-and-own with systematic reuse for developing software variants. In: *2014 IEEE International Conference on Software Maintenance and Evolution*. pp 391–400
- Shatnawi A, Seriai A, Sahraoui H (2015) Recovering architectural variability of a family of product variants. In: *Software Reuse for Dynamic Systems in the Cloud and Beyond vol. 8919 of Lecture Notes in Computer Science*. Springer International Publishing, pp 17–33
- van Solingen R, Basili V, Caldiera G, Rombach HD (2002) *Goal Question Metric (GQM) approach*. Wiley
- Rodgers JL, Nicewander WA (1988) Thirteen ways to look at the correlation coefficient. *Am Stat* 42(1):59–66
- Petersen W (2004) A set-theoretical approach for the induction of inheritance hierarchies. *Electr Notes Theoret Comput Sci* 53:296–308
- Shatnawi A, Seriai AD, Sahraoui H (2017) Recovering software product line architecture of a family of object-oriented product variants. *J Syst Softw* 131(C):325–346. <https://doi.org/10.1016/j.jss.2016.07.039>
- Linsbauer L, Lopez-Herrejon RE, Egyed A (2017) Variability extraction and modeling for product variants. *Softw Syst Model* 16(4):1179–1199. <https://doi.org/10.1007/s10270-015-0512-y>

**Publisher's Note**

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.